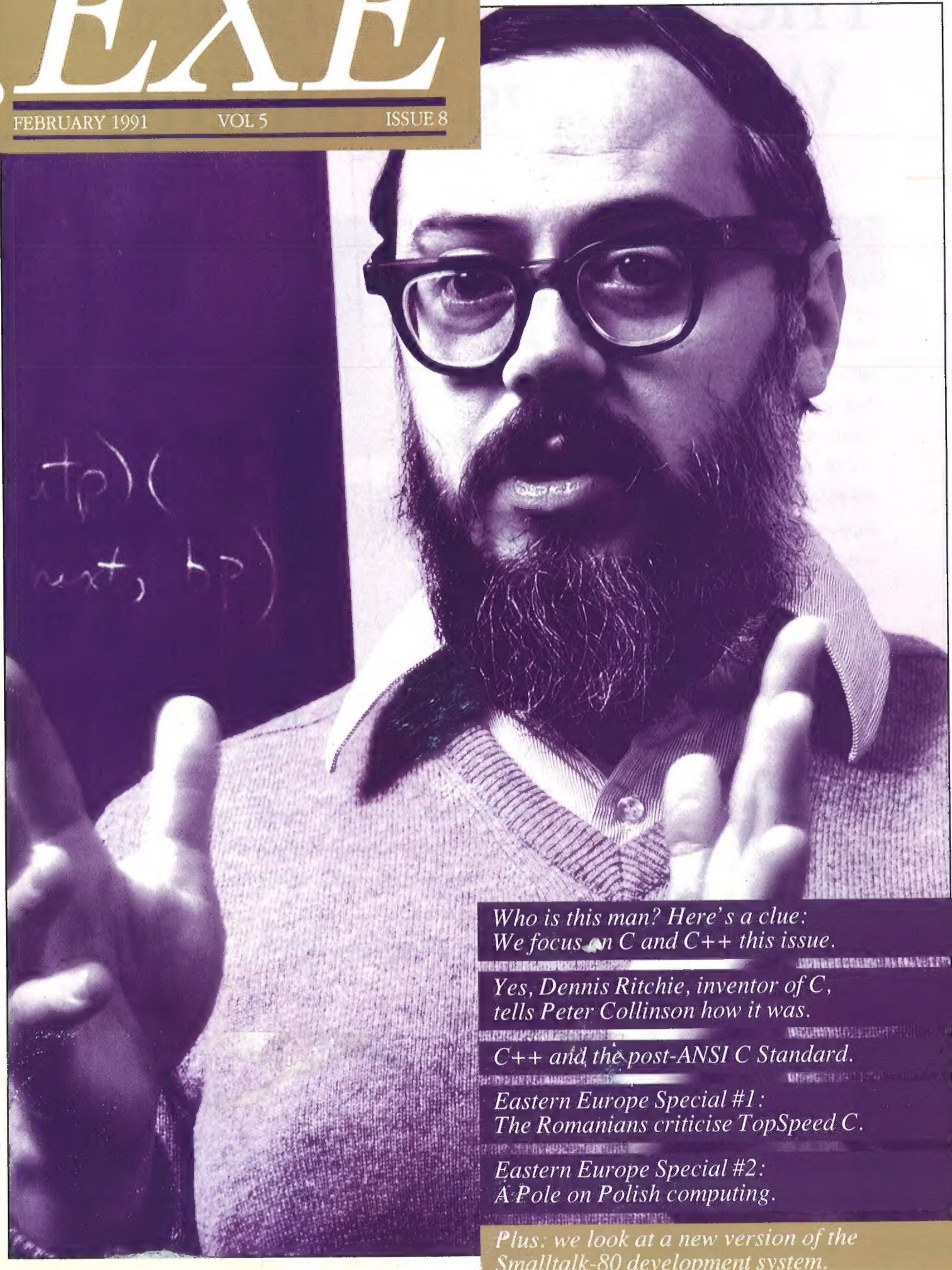
5th BIRTHDAY ISSUE

# .EXE

The Software Developers' Magazine

FEBRUARY 1991 VOL 5 ISSUE 8




*Who is this man? Here's a clue: We focus on C and C++ this issue.*

*Yes, Dennis Ritchie, inventor of C, tells Peter Collinson how it was.*

*C++ and the post-ANSI C Standard.*

*Eastern Europe Special #1: The Romanians criticise TopSpeed C.*

*Eastern Europe Special #2: A Pole on Polish computing.*

*Plus: we look at a new version of the Smalltalk-80 development system.*

# The Leading Light in Windows Training

QA

Training has long been recognised as the leader in Windows Training. The world's first company to offer Windows programming courses. The first authorised by Microsoft. And the very first to give technical training in Windows 3.

Our unique range of courses – Windows training for developers, support personnel and users – are widely acknowledged to be, quite simply, the best.

We also have worldwide involvement in consulting on many of the major Windows projects with corporate developers and software suppliers.

Only QA has this breadth of experience in every aspect of the Windows environment. Experience which is your assurance that a QA Windows course, is certain to maximise your skills and effectiveness.

*For more information, please complete and return the coupon today or call Samantha Trinder on 0285 655888.*

### Introduction to Windows

Use Windows mouse and keyboard controls; Customise the Windows desktop; Work with Windows file and print utilities; Use Windows applications productively; Understand the theory of Windows. *2 Days.*

### Supporting Windows

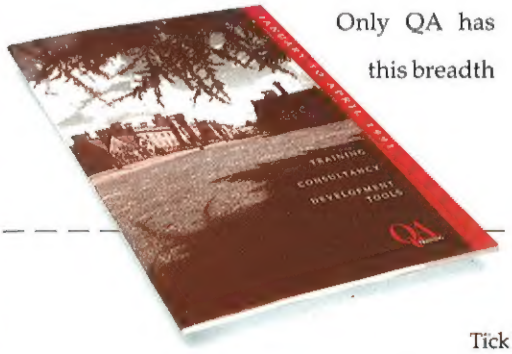
Install Windows and explore related issues; Configure Windows and applications; Use network, communications and print facilities; Understand Windows internals – memory, multi-tasking. *3 Days.*

### Microsoft Windows Programming

Use the MS Windows SDK and tools; Write well structured compatible Windows applications; Understand the Windows API and Windows programming principles. *5 Days.*

### Migrating Windows Applications to PM

Port Windows application to PM using the SMK; Understand structure and makeup of Windows and PM; Understand PM API including GPI; Understand object oriented structure for GUI code effectiveness; Aid migration with case oriented tools. *5 Days.*




Mail to QA Training Ltd, Cecily Hill Castle, Cirencester, Gloucestershire GL7 2EF. UK.

Tick here for full details:

☐ OS/2 ☐ Unix ☐ Windows
☐ C ☐ Object Oriented ☐ Networks
☐ Consultancy ☐ Products

NAME ________________ TITLE ________________

COMPANY ________________

ADDRESS ________________

________________ TEL. NO. ________________

QA TRAINING

The Leading Light in Windows Training

CIRCLE NO. 445

Vol 5 Issue 8

# .EXE

February 1991

**Editor:** Willie Watts
**News Editor:** Dan O'Brien
**Editorial Assistant:** Wendie Leek
**Design & Layout:** Katerina Adams, Mark English
**Display Advertising Manager:** Sandra Inniss-Palmer
**Senior Ad Executive:** Ed Butcher
**Reader Services & Promotions:** Helena Adams
**Publisher:** Jon Howell

## General

.EXE Magazine is independent and not affiliated to any vendor of hardware, software or services. It is published by Process Communications Ltd, 10 Barley Mow Passage, Chiswick, London W4 4PH.
Tel: (Advertising/Editorial/Production) 081 994 6477
(Subscriptions) 044282 4501
Facsimile: 081 994 1533 Telex: 8811418 SPACES G
ISSN: 0268-6872

## Subscriptions

.EXE Magazine is a monthly journal for software developers. It is available only by subscription, at a cost of £35 per annum (11 issues) in the UK (see insert card for academic and overseas rate). 'A subscription implies that this journal will be sent to the subscriber until one of the three expires' - AG Macdonell. The magazine is published around the 26th of the month preceding the cover date. There is no January issue. To subscribe, please call 044282 4501, or write to The Subscriptions Manager, .EXE Magazine, 10 Barley Mow Passage, Chiswick, London W4 4PH. We can invoice your company or take ACCESS and VISA cards. In case of a query about your subscription, please call 044282 4501.

Back issues are available at a cost of £3.50 each - call our Chiswick office (081 994 6477) for a list of issues still in print. We also offer a range of magazine binders, disk holders etc - please call for details.

## Editorial

Editorial enquiries should be addressed to The Editor, .EXE Magazine, 10 Barley Mow Passage, Chiswick, London W4 4PH. We welcome letters, opinions, suggestions and articles from our readers. If you are interested in contributing articles, please write to this office for a copy of our Contributors' Guide.

Information contained in .EXE is believed to be correct. If errors are found, we will endeavour to publish a clarification in the next available issue.

From time to time, we offer to copy the PC program code described in an article onto diskette for our readers. In this case, please send a blank, formatted disk with a self-addressed, prepaid mailer to the editorial address given above. We can copy both 5.25" and 3.5" disks.

The publishers can accept no liability for any consequences of using software distributed in this way.

## Advertising

If you are interested in advertising in this magazine, please write to the Display Advertising Manager/Recruitment Sales Manager (as appropriate), .EXE Magazine, 10 Barley Mow Passage, Chiswick, London W4 4PH, or call 081 994 6477 for details of our advertising rates.

## Pronunciation

The name of .EXE Magazine is pronounced to rhyme with 'not sexy magazine'.



## Special Issue - C and C++

# Don't Shoot the Programmer

*According to Ray Jones, it's the managers that need training in Structured Methods, not the programmers.*

It's unfortunate but true that even after the great progress made in structured methods of analysis and design, and CASE tools, programmers still do badly. Software is frequently delivered late and below spec. In suit-speak, we have an 'ongoing software backlog'.

Yet there can be no doubt that the methods and tools developed over the last few years are effective. All the evidence points to this. The adoption of structured methods in software development improves quality, eases maintenance and shortens time scales. And the proper use of CASE tools to support the methods can only improve matters. So why don't more people use the methods? Or rather, why don't they use them properly?

Hands up those of you who have been on structured methods courses - gosh! there are a lot of you. Now, hands up those of you who adopted those methods and are still using them six months after attending the course. Come on, be honest now. Aha! Rather fewer this time, I see. In fact, a lot fewer. Why is that? Why, when you were all fired up with enthusiasm at the end of the training course, did all your good intentions dribble away in the sand, like so many New Year's resolutions?

One of the reasons is that, when it comes to analysing the *real* requirements of *real* people and implementing *real* systems in the *real* world, it's not as easy as it seemed in the friendly and supportive atmosphere of the class-room, model solutions printed upside-down at the back. But then it never is. Nobody really expected it to be.

Another reason for the decline - and I believe it is the main one - is that when you get back to the office, the atmosphere is not friendly and supportive.

There you are, sketching elegant Data Flow Diagrams and exquisite Entity Relationship Diagrams, using your newly-learned skills to construct a Software Environment Model of the system that you will be implementing. But your project manager looks on, with the air of a Cromwellian watching them putting the undercoat on the ceiling of the Sistine Chapel.

'Where's the code, Sam?'

'Well, you see, with this diagram I will be able to plan the modules so that...'

'Where's the code, Sam?'

'Look. The data flows through the system starting here, where the invoices are entered by...'

'Where's the code, Sam?'

The project manager hasn't been on the Structured Methods course. He is too important to go on courses and anyway, he doesn't rate them after a disappointing punched card handling course he attended in 1975. He only understands one thing: that the result of your labour ought to be program code. He wants it now. Pretty diagrams might look impressive, but code is produced by sitting at a keyboard and typing.

So what do you do? The pressure is on. 'I need to be able to show some progress on this project at the next management meeting', says your manager. 'I need the first five modules coded by the end of the month'. Do you stick up for the methods of Gane and Sarson, Ward and Mellor or Michael Jackson? Do you quote the sayings of DeMarco or Boehme to support your case? Do you wave goodbye to the chance of an early bonus? Do you heckers. You extract your digit, and code the first five modules by the end of the month.

It seems to me that this is a serious problem, and that it must be addressed. Training the analysts and programmers is fine - no, it is essential. But managers must also be trained in the modern methods of software production. They should know that the code is the last thing to be produced and they should be aware of the necessary stages in the development process that precede the coding. Only in this way can project managers successfully plan and monitor the progress of their projects. And only in this way, with proper management participation, will sensible and efficient development methods really begin to catch on.

EXE

*Ray Jones is a Software Engineer who really does try to use proper design methods, honest.*

## THE C LANGUAGE

**Microsoft C V6 is a complete rewrite with improved optimisation and a new Programmer's Workbench**
**High C V1.6 has been considerably improved, with better Microsoft C compatibility, and new documentation**

### C COMPILERS

| | | |
|---|---|---|
| Avocet AVCOM 86C | PC-DOS | £210 |
| Aztec C86 Professional | MS-DOS | £120 |
| Aztec C86 Developer | MS-DOS | £175 |
| Aztec C86 Commercial | MS-DOS | £265 |
| CI C86 Plus v1.3 | MS-DOS | £290 |
| ECO-C88 V4.0 | MS-DOS | £ 70 |
| HIGH C (Metaware) v1.6 | MS-DOS | £380 |
| HIGH C 386 v2.3 | PL386&MS-DOS | £640 |
| Hi-Tech C | MS-DOS | £135 |
| Lattice C v6 | OS/2&MS-DOS | £175 |
| Let's C v4 | MS-DOS | £ 60 |
| Microsoft C v6 | OS/2&PC-DOS | £269 |
| Microsoft QuickC v2.5 | PC-DOS | £105 |
| M'soft QuickC/QuickAsm | PC-DOS | £105 |
| MIX C | MS-DOS | £ 20 |
| MIX Power C & lib s'ce | PC-DOS | £ 29 |
| Turbo C++ | PC-DOS | £120 |
| Topspeed C for OS/2 | OS/2 | £335 |
| Topspeed C (std) | PC-DOS | £135 |
| Topspeed C Extended | PC-DOS | £270 |
| Watcom C v8 std | MS-DOS | £269 |
| Watcom C v8 Prof | OS/2&DOS | £335 |
| Watcom C/386 std | PL386&MS-DOS | £625 |
| Watcom C/386 Prof | PL386&MS-DOS | £830 |
| Zortech C++ v2.1 | PC-DOS | £120 |
| Aztec C86 Developer | CP/M-86 | £245 |
| Aztec C86 Personal | CP/M-86 | £140 |
| Hi-Tech C | CP/M-86 | £150 |
| Lattice C | CDOS | £380 |
| Avocet C | CP/M-80 | £210 |
| Aztec C Personal 1.06D | CP/M-80 | £120 |
| Aztec Commercial 1.06D | CP/M-80 | £200 |
| Hi-Tech C | Z80 + CP/M-80 | £ 99 |
| Mix C | Z80 + CP/M-80 | £ 35 |
| Aztec C65 V1.05 | Apple DOS | £200 |
| Aztec C68K MPW C | MACINTOSH | £110 |
| Aztec C68K C | MACINTOSH | £ 75 |
| Aztec C68K/AM Developer | AMIGA | £175 |
| Aztec C68K/AM Prof'snl | AMIGA | £120 |
| Aztec C68K/ST Developer | ATARI | £175 |
| Aztec C68K/ST Prof'snl | ATARI | £120 |
| Hi-Tech C | ATARI | £ 99 |
| Mark Williams C v2 | ATARI | £110 |
| Laser C | ATARI | £135 |
| Prospero C | ATARI | £105 |

### C CROSS COMPILERS

**We supply 2500AD, Avocet, Aztec, Lattice, IAR and Hi-Tech Cross Compilers hosted on MS-DOS and targeted on Z80, 6502, 6801, 68HC11, 6301, 6809, 7811, 8051, 8096, 68000 & 68020. Please call for information or advice.**

### C INTERPRETERS

**Instant C/16M supports large programs on 286/386 machines.**

| | | |
|---|---|---|
| C-terp V3.0X | PC-DOS | £165 |
| Instant C/16M | PC-DOS | £700 |
| Interactive C | PC-DOS | £195 |
| Introducing C | PC-DOS | £ 85 |
| Living C Plus | PC-DOS | £135 |
| RUN/C Professional | MS-DOS | £110 |

## FORTRAN COMPILERS

**The latest Prospero Fortran now supports OS/2 & DOS.**
**A new version of Lahey's compiler with improved optimisation now here.**

| | | |
|---|---|---|
| Lahey F77L v4.10 | MS-DOS | £410 |
| Lahey F77L-EM/32v3 ergo+ | PC-DOS | £665 |
| Lahey Personal Fort-77 | PC-DOS | £ 79 |
| FTN77/386 Fortran | PC-DOS | £765 |
| RM/FORTRAN 77 v2.43 | MS-DOS | £445 |
| MS-FORTRAN 77 v5.0 | OS/2&MS-DOS | £250 |
| Prospero Fort 2.10 | OS/2&MS-DOS | £240 |
| Pro Fortran for GEM | MS-DOS | £ 80 |
| Prospero PC Fortran | PC-DOS | £ 80 |
| Utah Fortran | MS-DOS | £ 35 |
| FS-Fort.(CGA & Herc) | MS-DOS | £ 34 |
| Watfor 77 | PC-DOS | £290 |

## PROGRAMMING TOOLS

| | |
|---|---|
| Ada Compilers | Algol Compilers |
| Assemblers & Libs | AWK |
| Basic Compilers | Basic Interpreters |
| Basic Utilities | Basic Libraries |
| BCPL Compilers | C Compilers |
| C Interpreters | C Libraries |
| C Utilities | Cobol Compilers |
| Comms.Libraries | Cross Assemblers |
| Database Libs. | Debuggers |
| Dis-assemblers | Editors |
| Engineers Libs. | Expert Systems |
| Forth | Fortran Compilers |
| Fortran Libraries | Graphics Libraries |
| Icon | Linkers/Locaters |
| Lisp | Modula-2 |
| Nial Interpreters | OPS 5 |
| Pascal Compilers | Pascal Libraries |
| Prolog | Rexx |
| Risc | Screen Libraries |
| Smalltalk | Snobol |

We stock many items for which there is no space in these advertisements.

## C++ COMPILERS

| | | |
|---|---|---|
| Guidelines C++ | PC-DOS | £370 |
| Turbo C++ | PC-DOS | £120 |
| Turbo C++ Prof | PC-DOS | £200 |
| Zortech C++ | PC-DOS | £120 |
| Zortech C++ Dev Edition | PC-DOS | £270 |
| Zortech C++ OS/2 Dev Ed | OS/2 | £360 |
| Zortech C++ 386 Dev Ed | PC-DOS | £540 |

### LIBRARIES

| | | |
|---|---|---|
| C++/Views (ZC+) | PC-DOS | £270 |
| Greenleaf Comms (T&ZC++) | PC-DOS | £195 |
| Object Vision | PC-DOS | £325 |
| Zortech C++ Database | PC-DOS | £180 |
| Zinc Interface Lib | PC-DOS | £150 |

## DISK COPYING SERVICE

**We can copy files to and from 600 disk formats including CP/M, CP/M-86, MS- DOS, PC-DOS, ISIS, APPLE, SIRIUS, BBC, TORCH, APRICOT, HP-150, TRSDOS, DEC RT- 11, IBM BEF, ATARI ST, AMSTRAD, MACINTOSH.**

**Our charge is £10.00 + disk + VAT with discounts on small quantities and disks are normally despatched within 24hrs of receipt. For more information call us.**

## PRICES & DELIVERY

**Prices do not include VAT or other local taxes but do include delivery in UK and Europe. Please check prices at time of order, ads are prepared some weeks before publication.**

**This page lists some of our products. Call us for a complete pricelist.**

**Order by phone with your credit card.**

## ASSEMBLERS

| | | |
|---|---|---|
| 2500AD 80386 ASM v5 | PC-DOS | £225 |
| MS Macro-86 v5.1 | OS/2&PC-DOS | £ 95 |
| PharLap 386 DOS Ext SDK | MS-DOS | £370 |
| PharLap 386 ASM/Linkloc | MS-DOS | £890 |
| PharLap 386/VMM | MS-DOS | £240 |
| Optasm | MS-DOS | £105 |
| Turbo Debugger v2 | PC-DOS | £105 |
| 2500AD Z80 ASM | CP/M-80 | £ 70 |
| SLR Z80ASM | CP/M-80 | £ 40 |
| SLR Z80ASM-PLUS | CP/M-80 | £140 |
| SLR MAC | CP/M-80 | £ 40 |
| SLR MAC-PLUS | CP/M-80 | £140 |
| SLR 180 (Hitachi) | CP/M-80 | £ 40 |
| SLR 180-PLUS (Hitachi) | CP/M-80 | £140 |

**Not all assemblers are supplied with a linker. Check before ordering.**

## C LIBRARIES

### DATABASE/FILEHANDLING

| | | |
|---|---|---|
| Btrieve for OS/2 | OS/2 | £315 |
| Btrieve for Windows | MS-DOS | £315 |
| Btrieve v5.1 | MS-DOS | £315 |
| XQL | OS/2,PC-DOS | £415 |
| Xtrieve Plus | OS/2,PC-DOS | £315 |
| C/Database T'lchest sce | PC-DOS | £ 39 |
| CBTREE | (source any C) | £135 |
| C-Index/Plus | (source any C) | £175 |
| C-ISAM (L,MS) | MS-DOS | £215 |
| C-tree | (source any C) | £210 |
| C-tree Plus | PC-DOS | £350 |
| Essential Btree (s'ce) | PC-DOS | £120 |
| CQL s'ce | PC-DOS | £270 |
| Faircom Toolbox Prof | PC-DOS | £610 |
| Pro-C & Workbench v2 | PC-DOS | £585 |
| R-tree | MS-DOS | £160 |
| db-VISTA DBMS Object | PC-DOS | £450 |
| Lattice dBCIII (L,MS) | MS-DOS | £185 |
| Lattice dBCIII+ | OS/2&MS-DOS | £365 |
| SftFcus Btree&Isam | (sce any C) | £ 80 |

### GRAPHICS

| | | |
|---|---|---|
| Enhanced Graphics Tlkt | PC-DOS | £170 |
| Essential Graphics v3 | PC-DOS | £215 |
| GFX Graphics | PC-DOS | £ 80 |
| GraphiC v5 (CI,L,DS,MS) | PC-DOS | £290 |
| graphics MENU/META v3 | PC-DOS | £135 |
| GSS Kernel system | OS2,PC-DOS | £425 |
| GSS Graph Dev T'kit | OS2,PC-DOS | £425 |
| HALO v3 (MSC5,LAT,TC) | PC-DOS | £210 |
| HALO Professional (C) | PC-DOS | £335 |
| HGraph (MSC) | PC-DOS | £ 80 |
| Ingraf (MSC) | PC-DOS | £175 |
| MetaWINDOW v3.7b | PC-DOS | £150 |
| MetaWINDOW/Plus | PC-DOS | £200 |
| Professional Halo | PC-DOS | £335 |
| TurboWINDOW/C (Turbo) | PC-DOS | £ 85 |
| QuickWINDOW/C (QC) | PC-DOS | £ 85 |

### SCREEN & WINDOWS

| | | |
|---|---|---|
| Vitamin C for OS/2 | OS/2&DOS | £240 |
| Vitamin C (source) | PC-DOS | £165 |
| Power Screen (MS&TC) | PC-DOS | £105 |
| Blaise View Mngr.(s'ce) | PC-DOS | £245 |
| Lattice Curses | PC-DOS | £105 |
| Entelekon Windows(s'ce) | PC-DOS | £105 |
| Multi-windows (MS,L) | PC-DOS | £190 |
| Panel Plus II (source) | PC-DOS | £270 |
| Vermont Views (MSC,TC) | PC-DOS | £345 |
| Greenleaf Datawindow | PC-DOS | £210 |
| Greenleaf Makeform | PC-DOS | £ 70 |
| C-Scape(+s'ce&Look&Feel) | PC-DOS | £345 |
| Curses/PC w. s'ce | PC-DOS | £240 |

### GENERAL FUNCTIONS

| | | |
|---|---|---|
| C+O Class Lib (MS,TC) | PC-DOS | £155 |
| C+O Cl's Lb (MS)WINDOWS& | PC-DOS | £155 |
| C/Utilities T'chest sce | PC-DOS | £ 39 |
| C-Worthy v1.2 + forms | MS-DOS | £340 |
| CodeRunner | PC-DOS | £105 |
| PDK£1 | PC-DOS | £ 75 |
| Greenleaf Functions s'ce | PC-DOS | £125 |
| G'leaf Super Functions | PC-DOS | £160 |
| G'leaf Fin'cial Mathlib | PC-DOS | £210 |
| Lattice C Comp Kit+s'ce | PC-DOS | £160 |
| The Heap Expander(MS&TC) | PC-DOS | £ 65 |
| C Tools Plus/6.0 (MS) | PC-DOS | £105 |
| Power Search | PC-DOS | £105 |
| Turbo C Tools v2 | PC-DOS | £105 |
| Essential C Utils s'ce | PC-DOS | £145 |
| Entelekon Funct(source) | PC-DOS | £105 |
| Resident-C (MS,L)(s'ce) | PC-DOS | £145 |
| Silverware C EMM Lib | PC-DOS | £115 |
| WKS Library | PC-DOS | £160 |

### COMMS LIBRARIES

| | | |
|---|---|---|
| C Asynch Manager V3s'ce | PC-DOS | £130 |
| Essential Comms(L,MS,T) | PC-DOS | £175 |
| Greenleaf Comms(source) | PC-DOS | £195 |
| Multi Comms (L,MS) | PC-DOS | £190 |
| Silvercomm C Asynch | PC-DOS | £145 |

### SCIENTIFIC LIBRARIES

| | | |
|---|---|---|
| C Lang Sci Lib v2 | ANY C | £160 |
| Mathpak 87 (L,MS) | MS-DOS | £ 80 |
| Science & Eng Tools | MS-DOS | £ 59 |

### PROGRAMMERS UTILITIES

| | | |
|---|---|---|
| PC-Lint | OS/2,MS-DOS | £ 85 |
| C-Documentor | PC-DOS | £195 |
| C-Scan | PC-DOS | £195 |
| Clear+ for C | PC-DOS | £140 |
| Lattice Comp Companion | MS-DOS | £ 65 |

### REAL TIME & MULTI-TASKING TOOLS

| | | |
|---|---|---|
| Concurrent C (PC/MPX)sce | MS-DOS | £ 60 |
| Multi C (L,MS,CI,TC) | PC-DOS | £190 |
| Op.Sys.T'box (MS) s'ce | PC-DOS | £ 69 |
| Timeslicer v5 (MS5) | PC-DOS | £165 |
| Over-C (L,MS) | PC-DOS | £225 |

## CROSS ASSEMBLERS

**We supply cross-assemblers by Avocet, 2500AD,Crash Barrier (METAi),IAR Systems and Pecan hosted on MS-DOS, CP/M-86 and CP/M-80 with over 30 target processors. In total over 200 products with no space to list them here. We hold some stock but you should allow 10-14 days for delivery.**

# GREY MATTER

*2 Prigg Meadow, Ashburton, Devon TQ13 7DF*

***TEL. (0364) 53499***


# GREY MATTER

*2 Prigg Meadow, Ashburton, Devon TQ13 7DF*

***TEL. (0364) 53499***

# GREY MATTER

*2 Prigg Meadow, Ashburton, Devon TQ13 7DF*

***TEL. (0364) 53499***


CIRCLE NO. 446

### Debugger for NDP Compilers

*Microway has launched a symbolic debugger for its NDP set of compilers for the 386/486. All the usual functions are included - source code display, single stepping, variable watching - together with support for 386 hardware debugging registers, and Intel and Weitek co-processors. The price is £250; Microway is on 081 541 5466.*

### BPS drops run-time licence

*Bits Per Second Ltd has upgraded its Graphics Server product, mentioned here in November last year. V1.1 brings various technical improvements, such as the ability to open a Graphics Server window as a child window of an application, and a new pricing policy: BPS no longer charges for run-time copies of the product incorporated into applications. Graphics Server costs £275; the details are on 0273 727119.*

### Fox wins Ashton-Tate lawsuit

*Ashton-Tate's copyright lawsuit against Fox Software, a manufacturer of a dBASE-compatible product, has been dismissed in a Californian court. The judge found that A-T had deliberately concealed the fact that dBASE was derived from a public domain program developed at the Jet Propulsion Laboratory. This deception was deemed to invalidate A-T's copyrights on dBASE. This result has interesting implications for the many other look-and-feel type lawsuits currently under way; in many cases the plaintiff is suing to protect his rights to software ideas which he himself had originally copied from a third party.*

### MS-DOS in ROM

*Ever fancied your own personalised MS-DOS chip? Microsoft has released a developer's kit for the ROM version of MS-DOS, including a mini-licensing agreement to make up to five copies of the generated code. The pack consists of a kit for configuring and adapting the ROM image, reference manual, and documented IO.SYS source code. The system is aimed at small scale embedded-system OEMs, and can be ordered on 0734 391 123.*

### OS/2 Memory Utility

*Masochists will love the new utility for PM from the OS/2 User Group. It displays in graphic form the amount of memory used by your OS/2 system. It's called Memmon (a pun on that other beast with an endless appetite, presumably) and costs £75 to members, £95 to others. Details available from Susie Hubber on 0285 655888.*

## Intel development kit puts Phar Lap's nose out of joint

Intel has taken the plunge into the software development market with a new 32-bit C compiler. The Intel 386/486 C Code Builder Kit contains a compiler, a source-level debugger, linker, libraries, utilities, MAKE and DOS Extender. The latter conforms to the fledgeling DOS Protected Mode Interface (DPMI) standard, and provides the user with full access to the full 4 GB address of the 386 processor. Significantly, there is no royalty for distribution on derived products. The package is not yet available in the UK, but retails in the US for $695.

Phar Lap, the software house which made its name with DOS Extender technology, was less than chuffed by the announcement. Phar Lap chief Richard Smith told me: 'It seems strange that Intel, a chip company, is moving into the software market. One moment we're allies, all working together to produce the best possible tools to use the chips; the next thing we know, we are in competition with each other.' Regarding the DPMI standard, which is largely an Intel product, Smith noted that Intel had done a good job of bringing all the software developers together, but admitted that Intel's intentions in the software development market were not known at the time.

Commenting on a recent *Guardian* article which suggested that the introduction of DPMI servers would put DOS Extender producers out of business, Smith said: 'This just shows a misunderstanding of what we do. We are in the business of producing general development tools, rather than just DOS Extenders. The DPMI server represents just one quarter of what the developer uses; he'll still need our products to write his programs.'

You can get hold of Intel in the UK on 0793 696000. Contact Phar Lap in the US on 0101 617 661 1510.

## Copyright Clamp-down

Several events in the last two months have signalled that companies throughout the industry are taking a tougher stance against piracy.

In December, the three largest UNIX vendors in the UK, Informix, SCO and Uniplex, combined forces with the pressure group FAST to combat what they describe as 'dangerous' levels of software theft amongst UNIX resellers.

Apparently, the bent dealers copy purchased software onto customers' systems, and then walked out with the same disks. FAST claims that up to 50% of current UNIX software is illegally installed in this way. The campaign to stamp it out will include a nationwide advertisement campaign aimed at resellers, and the publication of a safety check-list for buyers.

Meanwhile, in DOS-land, the Business Software Alliance, a group representing six of the largest PC software houses, has announced that four of their number are to file legal proceedings against Marconi Instruments Ltd for alleged software theft. The companies - Ashton-Tate, Lotus, Microsoft and WordPerfect - obtained court authorisation for an 'Anton Piller' order on the electronics firm in December. Under English law, this allows a solicitor to inspect without advance notice the premises of the company and obtain evidence which might otherwise be destroyed. Since the collection of evidence has always been the primary stumbling block of software theft prosecutions, this case, if successful, could well lead to further prosecutions.

At the same time as the Marconi prosecution was taking place, another two companies, Rhone-Poulenc Films and France Distribution Systems, were facing similar BSA-initiated litigation in France. The BSA is taking a particularly tough line in Europe, where software piracy is perceived to be more of a threat than in the States: a belief largely derived from national comparisons of the ratio of computer programs licensed to the number of machines sold. In America, the figure is 1.88 packages per machine - Europe averages around 0.6.

## Dual monitor PS/2

Pity the poor PS/2 owner who wants to run CodeView in dual monitor mode. If he is determined to proceed, he must fork out for an IBM 8514/A video adapter (or perhaps an XGA would do the trick?) with hugely expensive high-definition monitor. Bournemouth-based Cebra Communications Ltd offers, for £495, a micro channel VGA board which cohabits with the one on the motherboard. As well as driver software to solve the CodeView problem, the company also bundles a Windows 3 driver. You can use this to perform that wonderful Mac trick of placing two monitors side-by-side and being able to drag a window off the edge of one, across the gap and onto the other.

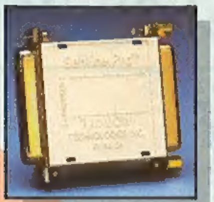
Cebra has lots of other interesting video adapter cards for both traditional XT/AT bus machines and MCA - with enough of them you do things like hanging nine separate video displays off one PC. You can call the company on 0202 299048.

SOFTWARE SECURITY

# With £1 Billion Worth Of Protected Software...

# Rainbow Is The Safe Bet.

*SentinelPro™*



*• Runs under DOS, OS/2 and Xenix • Algorithm technique (Never a fixed response) • External parallel port installation • Minimal implementation effort • Higher level language interfaces included • 100 times faster than fixed-response devices (1 ms) • ASIC design for reliability*

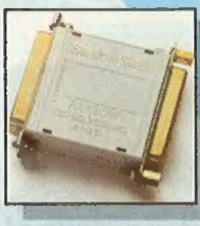
*Sentinel-C™*

*• Protects multiple packages with one device • 126 bytes of non-volatile memory programmed before shipment of the software • Rainbow supplies a unique adapter for programming the unit • Higher level language interfaces included • Runs under DOS, OS/2 and Xenix • External parallel port installation*

*Eve™*

*• For the Macintosh SE and II • Complies with Apple Desktop Bus Interface requirements • Rainbow-assigned developer passwords to prevent tampering by other developers or sophisticated "hackers" • 7 locks per key, usable individually or in combination, on one or up to seven applications*

*SentinelShell™*



*• Runs under DOS on IBM PCs and compatibles • Protects without requiring access to the source code • Completely transparent to the end user • User-friendly software • Pocket-size key attaches quickly to any standard PC parallel port • ASIC design for reliability*

*DataSentry™*

*• Completely user-installable • Pocket-sized external device • Menu-driven, user-friendly interface • Single- or multi-user security system • Audit trail, log-on identifiers and automatic encryption/decryption of entire directories • Secures data transmitted by modems • Prevents recovery of data by utility programs*

Software developers creating the latest applications for the IBM PC/XT/AT, PS/2 and compatible systems can now turn to the Software Sentinel range of hardware keys for the *first-class* in *world-class* software protection.

There's the best-selling SentinelPro, known worldwide for its virtually unbreakable security, its ASIC technology and its invisible operation.

A close relation, the Sentinel-C for custom configurations, enables multiple package protection with a single device.

For the Apple market, security-minded Mac software developers can now secure their return-on-investment, too. Eve plugs into the Mac ADB connector and is completely transparent to the user–providing up to seven programmable security locks per key.

Rainbow's latest protection strategy is the SentinelShell–that lets users place a "shell" around existing, off-the-shelf programs. Access can be limited to those issued a key, so that libraries, educational establishments and corporations can very simply guard their investments.

Available soon from Rainbow is the DataSentry, a user-installed key that provides low cost security for sensitive data in both database applications and corporate/banking environments.

No matter where you sell your software worldwide, stay in control of your distribution and revenue by choosing the internationally accepted standard in protection... Rainbow Technologies. Be sure. Protect your pot of gold at the end of the rainbow.

## CALL TODAY OR ORDER NOW.

☐ Please send me a SentinelPro Evaluation Kit.
I enclose £50 + VAT Payable to Rainbow Technologies Ltd.

☐ Please debit my credit card. Access ___ Visa ___ Amex ___
Expiry Date ______ (PLEASE TICK)

☐ Please send me more information.

Name ______

Position ______

Address ______

Telephone ______

Signature ______ .EXE 2/91

RAINBOW TECHNOLOGIES LTD., Shirley Lodge, 470 London Road, Slough, Berkshire SL3 8QY.

**RAINBOW TECHNOLOGIES**

Shirley Lodge, 470 London Road
Slough, Berkshire SL3 8QY
Tel: 0753-41512 Fax: 0753-43610



CIRCLE NO. 447

## Stats News

A new product from those Top Maths Boffins at NAG, who have released a graphics package and prototype for their industry standard maths libraries. NAG is a non-profit academic organisation with a reputation for high quality mathematical software. They've left graphics alone until now, however. The new package is called PV-WAVE:NAG, and is a collaboration with the American firm Precision Visuals. It's only available on the Sun-3 and 4 workstations and VMS (why do we keep mentioning VMS this month?) at the moment, although a PC version is being mooted. More details from NAG on 0865 511245.

## Zortech broadens its range

Zortech Ltd has introduced a range of products based on its new 32-bit C++ compiler engine. The DOS 386 developer's edition contains a full 32-bit compiler, together with specially adapted 386 versions of the Zortech debugger, workbench (including an editor with a multi-gigabyte capacity), a standard library with source, a set of tools and the Flash Graphics library. The package is based on Phar Lap's 386/DOS extender; the SDK for this needs to be purchased separately. The DOS 386 version joins the existing 'C++ for MS-DOS and Windows' and 'OS/2 C++ Developer's' products, making Zortech's offerings for the single-user PC pretty comprehensive.

Also due for release by the time you read this is C++ for SCO UNIX. As well as the compiler itself, Zortech has ported its workbench and Flash Graphics so that they can be used on the main console of SCO UNIX machines, and it is now working on getting the debugger to work in the same way. Other flavours of UNIX are also planned for the near future.

As if this wasn't enough, Zortech now has its eye on the Macintosh. C++ for the Mac is currently in Beta-test, with a planned release date in late March. This product will integrate with the Macintosh Programmers Workbench and the MPW debugger, so can be used as a direct replacement for MPW C and C++.

Prices: the DOS 386 compiler costs £599.95. To this you must add the Phar Lap kit, which Zortech sells at £299.95. The SCO UNIX package is also priced at a shilling under £300. Contact Zortech on 081 316 7777.

## Objects for Windows

Microsoft has released a preliminary specification and code of a new set of DLLs designed to support object-based communication between Window applications. The approach taken by the new system will be an extension of the current DDE technique of allowing many applications to control different elements within a single document. The Object Linking and Embedding specification (Microsoft wants it pronounced '¡Olé!') will be included in standard Windows in the next release and will also be implemented on the Macintosh System 7.0 when it arrives. It has the support of Aldus and Lotus among others. The latest versions of Microsoft Excel and Lotus Notes are the first two applications to use the new system. Copies of the specification are available from Microsoft on 0734 391123.

## Object Management Group Publishes Guide

The Object Management Group, the OOP Standards group, whose patrons include AT&T, Apple, Borland, Whitewater and HP, has released its Object Management Architecture Guide, a proposed standard for Object Oriented environments. The guide steers clear of specifics, but does present important descriptions of the OMG's proposed interfaces between differing systems, and a description of the OMG's evaluation procedures. OMG isn't a standard making body yet, and the omission of IBM and Microsoft from its ranks implies that it might never be, but the publishing of these guidelines has partly shaken off OMG's talking-shop image. Both Apple and Dataflex have joined the group following the guide's appearance. Copies of the £30 document can be obtained from OMG on 0332 513855.

## CASE:W for Glockenspiel CommonView

A new version of the popular application generator tool for Windows, CASE:W, has been announced, this time for users of Glockenspiel C++ and the CommonView class library. Programmers can now produce code in the familiar Notifier/Control/View class style. CASE:W for CommonView shares the same interface and features of the rest of the CaseWorks series, including the useful regeneration feature, which allows CASE:W source output to be hand tweaked, and then re-incorporated into the system.

CASE:W itself can run on a 'minimal Windows system', but it will, of course, need the Glock kit and the Windows SDK to compile. It costs £795, and is available in the UK from Systems Resources, Tel: 0203 220246.

**IBM takes Smalltalk/V**

*IBM has licensed Digitalk's Smalltalk/V for OS/2. Digitalk will be working with IBM on Smalltalk/V enhancements, including a set of visual programming tools, increased AD/Cycle integration and SAA compliance. A Windows 3 version of Smalltalk/V should be available now. Digitalk's products are distributed in the UK by Cocking and Drury (071 436 9481).*

**Assembly Offer**

*The Software Construction Company is running a promotion this month (February). Anyone purchasing the popular, but pricey £255 Spontaneous Assembly code library from TSCC will also receive a free copy of MASM V5.1 (usually retails for £115) and a copy of Using Assembly Language from Que (£25). TSCC is on 0763 244114.*

**Coherent UNIX Update**

*Coherent V3.1 is the newest release of Mark Williams' cheap UNIX-compatible operating system. New features include COM3 and COM4 support, a RAM disk and a vi/ex clone called Elvis. Not in there yet is a C compiler that produces anything larger than small model programs, but they're working on it. Coherent V3.1 is still $99.95. Mark Williams can be contacted in the US on 0101 708 291 6700.*

**APL for Windows 3**

*STSC has a new version of its APL*PLUS package for the PC. Features in version 10 include multiple instances in Windows, new mouse and PostScript printer support, and, for those APLers who would stoop so low, cross-linking with Microsoft C 5.1, Turbo C V2.0 and Microsoft FORTRAN V5.0. APL*PLUS PC is distributed in the UK from Cocking and Drury, 071 436 9481. STSC itself is on 0753 831541.*

**Quintec Prolog**

*A good month for the dropping of runtime licences, this. Prolog-maker Quintec Systems (0865 791565) has announced that it will no longer charge a licence fee on applications created with Quintec Prolog. The package, currently at V2.0, is available for UNIX workstations and PC running Windows 3.0.*

**Inmos and Ready Systems**

*Ready Systems Corporation, producer of the VRTX real time operating system, has entered into partnership with transputer maker Inmos. The first fruit of the new relationship will be a port of VRTX32 onto current and next generation (H1) transputer families.*

**ideaList**

*ideaList is a free-format database for the PC from Blackwell Scientific Publications Ltd. According to its promotional literature, it can find all references to any string in the database within milliseconds - the technology is similar to that used to drive mainframe information systems and CD-ROM databases. The package costs £225, ring 0865 240401 for the details.*

**Ada Still Free**

*Thornbrook, whose offer of a free Sun or VMS Ada Compiler suite to deserving academic institutions was mentioned in December's issue, has been so impressed with the interest generated that it is extending its offer to the 31st of March. To recap, the offer includes a full compiler, run-time system, debugger and documentation, and would usually sell for $15,000 to $70,000. John MacGregor awaits your calls on 0993 831333.*

**C into AutoCAD**

*Autodesk Ltd has shipped Release 11 of the well-known AutoCAD drafting program. The new feature of particular interest to developers is the AutoCAD Development System, which provides an API through which third parties can hook in their own program modules, which must be written in Metaware High C 386. ADS supplements the earlier AutoLISP system. AutoCAD costs £2500; the Advanced Modelling Extension is £400 extra. Autodesk may be contacted on 0483 303322.*

**Documentation Standard**

*The BSI has published a draft British Standard to cover the area of software documentation. Document number 90/66493 DC, 'Recommendations for the design of user documentation for software products for text and office systems', is intended to guide technical writers as to how to present information in a most effective way. The draft standard costs £6.25 to BSI subscribing members and £12.50 to others. The BSI is on 0908 220022.*

**Born out of need?**

*Milspec Systems Ltd, the Ada compiler company, has produced a RAM manipulating program. PLUS96K works by stealing 96 KB of graphics RAM from a PC's EGA/VGA board, increasing available 'conventional memory' from 640 KB to 730 KB. Naturally it doesn't work with graphics programs; but at £29.99 including VAT and postage it seems quite reasonably priced. Milspec's number is 0203 670770.*

## A new look for Software Tools

Software Tools '91, the trade show which caters for the software developer, is to be held at the Wembley Exhibition and Conference Centre from 11th-13th of June. This year, for the first time, it is to be sponsored by .EXE Magazine.

It has already been noted elsewhere that computer trade shows seem to be growing more and more dreary as time passes. Where once the aisles thronged with excited chatter and the crowds queued up to see the latest product unveiling; now a desultory trickle of bored, grey men wander around the lonely spaces, sometimes pausing at the odd booth to make a half-hearted attempt to strike a deal on the purchase of 30/off 386s.

We have determined that Software Tools '91 will buck this trend. There will be 'Spot the Bug' prize competitions, where punters riding a line of PCs will compete to get a bit of code back on its feet, and Software Marathons, where teams from the likes of Microsoft and Borland will race to complete set projects. There will be technical support booths, where you will be able to confront in person the bloke who is always so evasive on the phone. There will be lectures and presentations, covering all the latest GUIs and 32-bits and OOPs and OSs. There will be Really Important Gurus, who, their beards glistening in the conference hall lights, will clear up the deepest computing mysteries with a few well-chosen words. There will be a shoal of little stalls, where the software dealers will display their wares like fruit at Sainsbury's, and you can buy anything from a real time operating system or a sizzling C++ compiler to that interesting little utility that you've always thought would come in handy, but couldn't be bothered to send away for. The big software companies, the 'major players', will be there as well; parading their latest stuff and giving us glimpses of what is in the pipeline. And there will be lots of people: chatting with the vendors, trying out the software, discussing a few beers over pints of foaming algorithms.

At least, we hope that all these things will happen. We already have Borland, JPI, Microsoft and the Software Construction Company with us, which is a dandy start, and we have a few projects under way, but we need your help and support. If you would like to find out about booking space at Software Tools, you should contact Julia Dunford at Blenheim Online Ltd (081 868 4466). If you have a cracking idea for something we should do at the Show, please contact us at the .EXE editorial office (but be quick - time is short). And, of course, we hope to see you on the day.

## Protection for MegaBasic

MegaBasic is a PC-based version of that hallowed language. Developed in the US, it started life as North Star BASIC (remember the North Star Horizon?) and these days, according to Murdoch Mactaggart of UK distributors The Software Centre, finds many of its fans among ex-mainframe and mini programmers, typically working in engineering, process control and the like. The particular feature which attracts, according to Mactaggart, is the ability to get at low-level features; for example, it is possible to manipulate CPU registers directly.

A new version of the interpreter, called Extended MegaBasic (rather than Mega$^2$Basic, as you might have guessed), runs AT class machines or better in protected mode - thus gaining access to the 16 MB address space. The 80x86 segmented architecture restricts string variables to 64 KB; but string arrays can now be dimensioned up to 1 MB in length and numeric arrays to 500 KB. The language is upwards-compatible with non-extended MegaBasic, and the interpreter contains its own DOS extender - no further purchases are necessary. License-free run-time modules enable you to sell on your applications at will.

The Extended MegaBasic development system retails at £375. If you want to compile your programs, you will need a copy of the standard MegaBasic compiler (£275) which, in a way that I don't fully understand, is able to produce both normal and protected-mode programs depending on which library you use. Phone The Software Centre on 0935 862 609 for the gen.

## dBASE IV for VMS

Ashton-Tate is now shipping the VAX version of dBASE IV V1.1. To launch the new product at Comdex, Ashton-Tate bravely offered to port any existing DOS application to VMS, UNIX or Macintosh, apparently successfully. As well as running as a DOS-code interpreter, it will also support VAX Rdb/VMS databases using NAS SQL. UK prices start at £969.00+VAT for a single user package and £4090.00+VAT for a four-user licence, and the distributor is P & P Micros and Technology Plc (0925 830404).

# Letters

*We welcome short letters on any subject that is relevant to software development. Please write to The Editor, .EXE Magazine, 10 Barley Mow Passage, Chiswick, London W4 4PH. Unless your letter is marked 'Not for Publication', it will be considered for inclusion on this page.*

## PS/2 troubles

Sir,

I would welcome any solutions or suggestions from your readers on a problem I have encountered with the 'protected mode' on 80286 and 80386 machines. I use the BIOS interrupt 15h function 89h to switch to protected mode, whereupon I search extended memory for data previously put there by calls to Microsoft's XMS driver, HIMEM.SYS. This driver allows MS-DOS programs access to extended memory. On a standard AT, this seems to work perfectly. However, on a PS/2 Model 50 (80286 machine, MCA), the same program hangs after a few calls to the protect-mode search routine. This happens even though I disable all interrupts while the program is in protected mode. Maybe someone out there knows something about either the MCA or the new BIOS that I don't?

*Joe Tierney*
*London NW1*

## Problem solver

Sir,

Disappointing isn't it? After half a century of technical progress, the IT industry is still discussing the relative merits of various programming languages.

If object-oriented programming (OOP) had been a panacea, SIMULA would have swept the board in 1967. Similarly, despite the greatest hype yet perpetrated, Fourth Generation Languages (4GLs) still fail to fulfil the promises made on their behalf. Worse, the computer education of our children concerns itself with teaching specific coding languages. Concentrating our attention on such issues avoids the point and amounts to sublimation activity.

Surely it is past time to recognise that our industry exists to solve problems. Everything is secondary to that aim.

It seems reasonable to me that, in order to provide a solution, I must first describe the problem I wish to solve. From the problem description I should then be in a position to derive a solution. Indeed, I wish to be able to derive a number of solutions, one for each potential implementation choice. Further, if the requirements of the problem change (as they will), I would expect to change the problem description and to maintain the solution by regenerating it.

What is needed then is a specification language in which I can write problem descriptions which themselves can be transformed into an implemented solution. Preferably, the transformation should be automatic. The coding language of the implementation is incidental and can be based entirely on convenience. A minimalist might say that I only need four operators, `ASSIGN`, `ADD`, `IF`, and `GOTO`. However, productivity will no doubt be improved by an ability to build a library of the descriptions of the small building blocks common to many problems. I should not expect such building blocks to approach the complexity of the whole problem, as the re-useability of a description is in inverse proportion to its complexity.

Where should I look for such a specification language? I do not expect to find a universal cure but I do expect the various system development methods to share common cause with me. In practice, I use the Jackson System Development (JSD) method with whatever system software and coding language proves convenient (most recently, Pascal, COBOL and ADA).

*PG Rule*
*Senior Software Engineering Consultant*
*LBMS PLC*

## Hybrid Greek

Sir,

'Sieve of Erastophenes'? (.EXE Nov '90, Page 8)
Eratosthenes was the maths.
Aristophanes was the plays.

*Mike Kinghan*
*Principal Analyst Programmer*
*AT&T ISTEL*
*Oxford*

## Is anybody out there?

Have .EXE readers lost the art of letter-writing? The flow of letters into our Editorial office has been most disappointing over the last few months, and we have now reached the point where Something Must Be Done.

Starting next month, and at enormous expense, the writer of the best (as judged by the Editor) letter that we print on these pages will be rewarded with an elegant, stylish .EXE T-shirt. Readers who belong to the Campaign Against T-shirts may nominate some other .EXE trinket of approximately the same value - an elegant, stylish magazine binder, an elegant, stylish disk holder or whatever; see your elegant, stylish .EXE Directory for details.

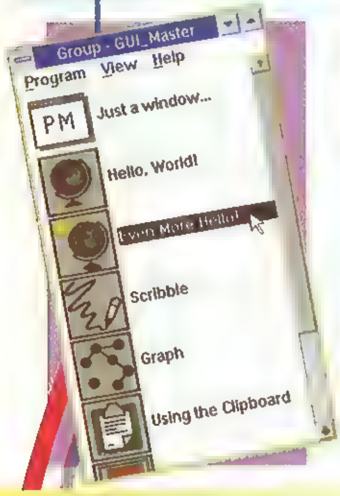
So if you have got an interesting coding problem, or you have discovered a plague of bugs in an expensive compiler, or you tried typing in one of our programs and it formatted your hard disk (only joking), or you think that one of our writers is an unrequited prat who should not be let within 200 yards of a computer, let alone allowed to try to *program* one, then reach down the Basildon Bond and drop us a line. We look forward to hearing from you.

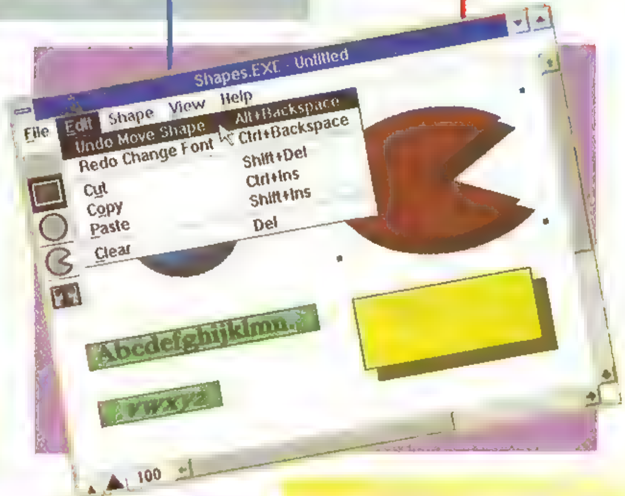
In the example programs you can see how *the GUI_MASTER* takes care of all the standard CUA functions like

- Scrolling
- Runtime error handling
- Printing
- Undoing/Redoing the most recent user actions
- Text editing
- Standard mouse interactions such as dragging and sketching
- Archiving objects
- Standard dialogs such as open, save as, fonts, color picker
- Using the clipboard

The 7 example programs range from simple to fairly complex. Together with the cookbook we provide they help you gain insight in all the possibilities of the *GUI_MASTER (Class Tree for C++)*. They come complete with fully annotated source code, so you can even use them as a basis for your own applications.

# Effectivity MS Windows Presentation




The *GUI_MASTER (Class Tree for C++)* enables you to specify different worlds, different representations of the same data which can be shown in different parts of the same window. If you modify some of the data in one world this is automatically reflected in the other world.

If you use *GUI_MASTER* your application is, as a matter of course, structured around the Data/World concept. An excellent basis for GUI applications.

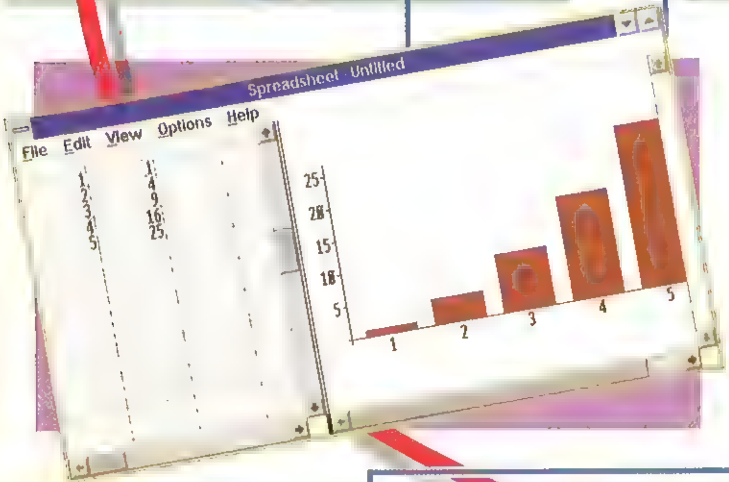
Anything that could be accomplished through 'normal' programming can be done if you use *GUI_MASTER*. It's just that *GUI_MASTER* makes it all a lot easier. Many of the things you might not include in your application because you think they're too complex or time-consuming to program, are easy to make with *GUI_MASTER*.

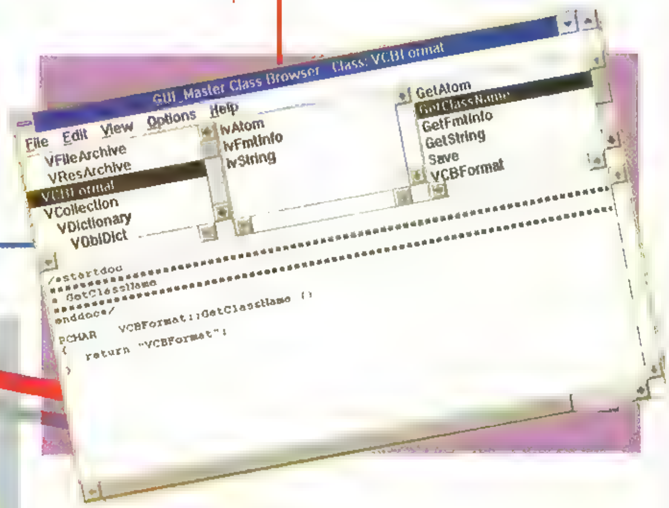
If you are building sophisticated C++ applications why not benefit from our *GUI_MASTER (Class Tree for C++)* to give you

- a head start in building GUI applications
- a GUI toolkit consisting of 85 reusable classes, together forming a well-tested and fully functional 'standard application'
- platform independence between MS/Windows and OS/2 Presentation Manager

The product offers

- 7 example programs
- a Class Tree
- a C++ Source Browser




The class browser is of crucial importance to enable you to view programs as collections of classes and not as a set of source files. It enables you to examine code made by others, and to understand the class structure of that code, thereby promoting the concept of 'the programmer as reader' and stimulating the reuse of software.

With our browser you can

- browse through the source code of any C++ program
- launch the editor of your choice to modify your C++ programs
- quickly locate any method in the Class Tree

# What Dennis says

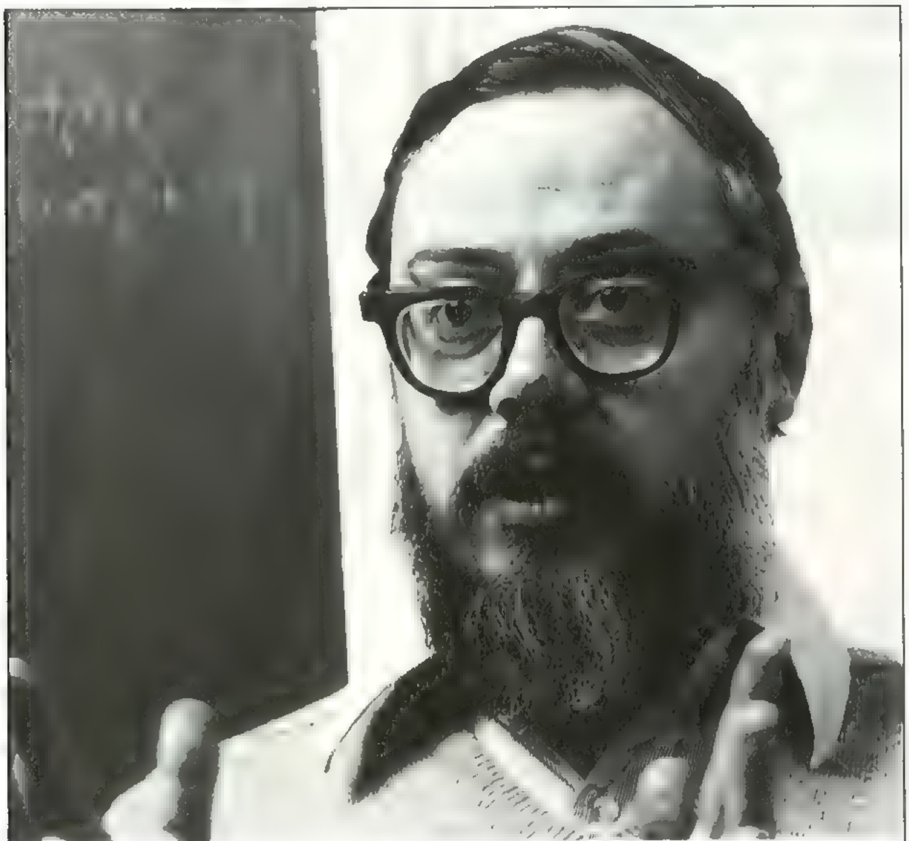
*If any programming language can be said to be the general-purpose standard, it is C. Peter Collinson has been talking to the man who invented it, Dennis Ritchie.*

***How did C start?***

On the PDP-7 UNIX system everything was written in assembly language. Doug McIlroy did an implementation of TMG, which is a compiler writing system done originally by Bob McClure. It's a top-down parser, although much more programmed than that. Doug McIlroy and Bob Morris did the original Multics PL/1 compiler in TMG.

***What was the main language of Multics, was that PL/1?***

It was PL/1. When we were on it, it was an early dialect of PL/1 called EPL that was essentially PL/1. Since we only had 4K words of memory for user programs on the PDP-7, PL/1 was clearly out of reach. Doug did TMG for the PDP-7. That was the year that Ken Thompson decided that we couldn't have a serious system without a FORTRAN compiler, so he sat down to write a FORTRAN compiler using TMG. This lasted about a day. Instead what happened was B. B was essentially a cut-down and modified version of BCPL.



*If you don't know what C is, then you have accidentally picked the wrong magazine out of the pile, and should put .EXE back at once. The C language was originally developed, together with the fledgeling UNIX operating system, at AT&T Bell Labs in the early 1970s. Dennis Ritchie is C's designer, and is also co-author (with Brian Kernighan) of 'The White Book' - The C Programming Language - which remains the definitive C reference. Peter Collinson interviewed him at Bell Labs, where he still works.*

***Had you been exposed to BCPL?***

At the time I came into the Labs, almost the very first thing that I did was to get hooked up with Rudd Canaday who was the person who had ported the Martin Richards' BCPL compiler from CTSS to the GECOS machine, the Honeywell GE-635. At the time he was moving the compiler to Multics. I got involved in Multics here by working on the BCPL compiler. This was an early dialect of BCPL; Martin Richards himself continued to change the language after we grabbed it.

BCPL had a strange way of doing external linkage, called the global vector. It didn't have some of the control statements. This is one of the reasons why `switch` statements in B and C differ from the BCPL `switchon` statement. I don't believe he had things like `endcase` at that time, so the languages differ in the way they come out of `case` blocks. I added `continue` later in B. Besides some obvious syntactic differences, some of the control structures were different just because it was an early version of BCPL.

BCPL was around and we had written some noticeable programs in it. On the first UNIX system, B was a new simpler language based on BCPL. It was an interpreter and didn't produce machine code; it produced an intermediate code.

***How did you bootstrap it, was it in assembler?***

It was first written in TMG, and later it was bootstrapped. In fact, the development of the language was amusing. The compiler was always pushing against the size limit in the machine. Ken would add something to the compiler and there would be a painful period when it was very hard to add something. But then he would include the new construct in the compiler - and that would make it smaller again allowing more new things to be added.

B was really the first high-level language that was used on UNIX. A few things were written in it. There were two implementations: one was this ordinary interpreter generating an intermediate stack based language. There was also a thing called `vb`, a Virtual B, that was a software paged version of the same thing allowing 4K word segments to be paged in and out by the interpreter. Whenever we had a program that got too big to fit into memory, `vb` was used.

By the time we got the PDP-11, only a few things had been rewritten in B. One of the first programs was the PDP-11 assembler. There was also a version of `dc`, the desk calculator which was the first program that ran on the PDP-11. It ran stand-alone before there was an operating system.

***Did you put in a standard DEC operating system on the PDP-11?***

We got the PDP-11 very early. It came during the summer of 1970. Only the processor and memory arrived, there was no disk. It was all paper tape software, you loaded programs with paper tape, there was no operating system as such. The first UNIX for it was written cross-assembling from the PDP-7 using the PDP-11 assembler that was written in B.

UNIX came up in two stages. Ken got it going before there was a disk. He divided the memory up into two chunks and got the operating system going in one piece and used the other piece for a sort of RAM disk. To try it out, you would load this paper tape that first initialised the disk and then load the operating system. There was a `cp`, a `cat`, and an `ls` actually running before there was a disk.

Once the system was really there, there was a regression. The B version of the assembler was pretty slow and that was rewritten in assembly language. I guess there were a few new things that were written in B. One of the earlier ones was the thing that did the expansion of stars and whatnot in filenames: the `glob` command. This stood for 'global' for reasons that escape me; it's not very sensible. The command did shell expansions. When the shell recognised it had a magic character in an argument, it would call this command to do the work. The code wasn't built into the shell.

## First came B...

***What was B like?***

Apart from tiny syntactic differences, B was exactly like C, except there was only one type that was *machine-word*. In that sense the semantics were the same as BCPL. There were no structures. The control structures were there. B did have some development of its own. I believe that the `for` statement got added somewhere in the middle of B development.

***He decided we couldn't have a serious system without FORTRAN, so he sat down to write a compiler***

C was developed by a process of evolution from B. The most important reason for change was that because of the typelessness of B, there was no really convenient way of directly dealing with bytes or even machine words. There really were two different kinds of things on the PDP-11, characters and integers. Even though the machine didn't have hardware for it yet, it was clear that there was about to be floating point and that would add at least two new different types. There would be one and two byte integers with four and eight byte floating point numbers.

The way that you deal with different types in BCPL is to have different operators. We had a version that did floating point and this was expressed by having different operators. Similarly there was a special function that was used for characters. This whole approach seemed to go OK on the word oriented machines, but it really broke down on the byte oriented machines.

The other thing was that it was fairly clear that once you began to compile code and produce decent machine language, the word orientation of the language was going to cause inefficiencies. The rule of the BCPL language is that you add one to a pointer to get to the next object, which is a word. On a byte oriented machine you are more or less forced to represent the pointers as counters of words, and to use them you have to shift them to produce the actual machine address. This seemed artificial. It was OK in the interpreter, it was glossed over as an interpretive overhead. But it was at least artificial and clearly would be inefficient.

The first thing for C was to introduce types to cope with these problems. This would allow efficient access to the hardware of the machine. Also, with the introduction of typed pointers, pointers contained addresses of an object of a type. Arithmetic on the pointers could be scaled appropriately.

Once the type structure was in C, I wanted to make a compiler so it would be fast enough to be able to be used for real things. At the time we were very much afraid of the compiler technology for the PDP-11, which was a two address machine that none of us had seen before. A single instruction, like a MOV instruction, could have two general operands. We were all used to single address machines, and for a while everybody was nervous about whether it would be easy to compile code for the machine.

I then heard about the PhD thesis of a guy who was working at Bell Labs at Indian Hill. The crucial idea of that was to have templates that matched bits of the parse tree. The templates have associated code fragments that would be inserted in the code stream. The code generators were written using this idea.

The two basic notions were: first, whether something is addressable. That is: whether or not some fragment of tree can be turned into an address mode instruction. Second, it was not addressable, whether or not it matches another table entry. If you have an assignment with operands that are both addressable, then you can produce a single MOV instruction.

There were four sub-tables: one for ordinary expressions that produce a value in a register; one for pushing a value into the stack; one for doing an expression only for its side effects, where you didn't care about the value; and finally, one for doing an expression in a conditional context where you generated a branch.

Anyway, that sort of worked.

## Structures

***Where did the idea of structures come from?***

I think that I added structures to versions of BCPL on Multics. I think that the Multics version had structures in the PL/1 style, namely with level numbers. That was fairly clumsy. The nested structures probably came from Pascal.

I wanted to add them to C. The other thing that went on at that time was a real resolution of the issue of arrays and pointers. To

go back a little: a vector or an array in BCPL was precisely a single cell containing a pointer to some space. If you said

```
let avec = vec 10;
```

`avec` became a pointer to the base of these 10 cells which were otherwise anonymous. In other words, the notion of the interchangeability of arrays and pointers that exists in C was even more explicit in BCPL. There really were no arrays, only pointers. The pointer was really in existence, and had a cell associated with it. You can add one to `avec` and it becomes a pointer to the next cell.

This view of things persisted into B, and was exactly the same in the early stages of C. The declaration of an array of 10 things produced a cell that had type 'pointer' internally and pointed to anonymous space. At this point, I began to think about how to put structures into the language. I realised that this notion didn't work with structures.

In BCPL there was an opportunity to create the pointer to the vector for both ordinary external static stuff and automatic variables. In the case of the structure, there was no point at which you would think that this pointer would get initialised properly. If you allocate a new structure containing an array, then exactly when does the array pointer get set up properly?

Similarly, structures were intended to be mapped onto memory. It seemed certain that if you were trying to describe some memory layout that included an array, then you would need some place to put the pointer.

It was at this point the crucial notion of C appeared that makes it different from almost any other language. Instead of having the array pointer to be explicit and live in storage associated with the array, the pointer should be created when you use it. In other words, there should be a conversion between the name representing an array and the pointer to the base of the array.

It turned out that with this idea, everything continued to work. All the programs had the same meaning as they had before, except that an explicitly defined cell became a pointer to the base of the array. It made things more efficient too because whereas before you were forced to pick up a pointer to do address arithmetic, now you could do things more directly. Also, it solved the structure problem. Suddenly, there was no instantiated pointer that we needed to fit into a structure or worry about how to initialise.

***In some ways, it was the structures that made C such a good assembler, if you see what I mean...***

Yes, that was our experience too, quite explicitly. In 1972, Ken [Thompson] sat down to write the operating system in C. At that point it was still in PDP-11 assembler. C did exist and did have a compiler by this time. He started, but he finally gave up. The first reason that he failed was that we was not able to work out the fundamental coroutine stuff, how the processes hand off to each other. That was just a conceptual problem, trying to work out the famous 'you are not expected to understand this' part of the code. That was just the first solution to that problem - but he hadn't even gotten that far at that stage.

***When we first got the PDP-11, you loaded all your programs with paper tape***

Perhaps equally important, there were no structures in the language. Therefore, representation of all the various tables - things like the file table, the inode table and the process table - were all quite artificial. He had to do all these using a BCPL technique: having a pointer and a bunch of constants, and subscripting using the constants which represented the fields in the table.

The trouble with that is that you cannot differentiate the types of the words. You can have a structure which has a character pointer, but all type information is lost when you use the base pointer plus subscript. So you cannot do anything sensible with the contents of words, and all the work on setting up types is wasted. Maybe you need to have parallel tables, one with all the characters and one with all the integers etc. All of this was a mess and just didn't work. By 1973, structures had been added and that was the spur that we needed. When we started again to write UNIX in C, the structures turned out to be very important.

***In some ways, it has taken quite a long time for structures to become full citizens.***

That is partly because the White Book definitely said that structure assignment is not in the language described by the book. On the other hand, it stated explicitly that structure assignment might be permitted in the future. Quite soon after the book appeared, the PDP-11 compiler had structure assignment and `pcc` had it. It became available and, although it was documented, it was never formalised in anything like a language description. It didn't always get into other people's implementations or there were slight differences in the implementation. For example, whether or not a function can return a structure that you can use a member of, like this:

```
val = (structfn()).member;
```

where `structfn()` returns a structure with member `member`.

If there had been a second edition of the White Book in 1980 or so, at least some of the confusion that lead to the ANSI committee wouldn't have occurred. Things like the question of unsigned characters, structure assignment, the unique structure names - member names within structures do not have to be unique across all structures, they only have to be unique within their own structure.

## Array problems

***I think that I have always found that somehow the history of how you invented arrays and structures has meant that people become confused. I mean the way that the address of a vector base is `vector` and also `&vector[0]`; and also that `vector[0]` and `*vector` return the same result.***

It does seem to be confusing. Perhaps that's because it was not explained correctly from the start. Although Brian Kernighan and I have had a chance to try to explain it, we may not have succeeded in making it obvious. Also, we haven't been helped by compilers, which have often been quite loose in what they allow. They often don't do anything like as strict enough type checking as they should, often not really checking the existing type structure of the language. So people get confused because they can try something which apparently works, and they are never told they have really broken the rules - that they are only being allowed to do this as an indulgence.

For example, this is how I feel about the business when `pcc` allowed ampersand to be placed in front of array names. It complained, but still allowed it. If the rules had been strictly insisted on by the compiler, so when people tried something they had immediate feedback saying 'this is wrong'; then they would have learnt the correct language a lot more easily.

Another problem is the business of rewriting the types of formal parameters to routines from 'array' to 'pointer' because it was just confusing. The specific reason that the rewriting is allowed had to do with the difference in notation at this moment when structures and the modern notion of arrays were both created. The original notation for a pointer declaration did not use stars, it used empty brackets, so instead of

```
*pp
```

you said:

```
pp[]
```

and that was indirection through a pointer. In the original language, an array was this cell containing a pointer to anonymous space. The notation for just the pointer was consistent with that. It was the pointer with no storage shown by the empty brackets. The survival of the bracketed parameters was a fossil of this notation.

But I agree that people do seem to have difficulty understanding all these things, and partly it may be inherent because it's an unfamiliar way of doing things, it may be an unnatural way of doing things although it is consistent. Partly it's that the language and its compilers allow a certain amount of slop that encourages this confusion.

***Do you feel ANSI got rid of any of this?***

Well, the trend in the last 10 years has been towards stricter checking of the rules that already exist. In structure references in the original language, you were allowed to use any pointer to the left of an arrow. The compiler simply said, 'OK, I don't care what type that pointer is, I'll believe what you have told me'. That had been tightened up by the time of the White Book, although not all the compilers enforced it.

ANSI has not actually made that much of a difference in terms of tightening up things, but they have certainly made the rules more explicit. But I cannot think of any aspect where they have made them stricter. In some sense they have made them looser because they put in the explicit `void *`, the universal pointer that you can coerce to and from without having to give an explicit cast. It's much better to do it explicitly, than happen all the time 'for free', as it did in the original. The original language used `char *` rather than `unsigned` and all sorts of nonsense like that.

There has definitely been a process of strengthening the type-checking, which is not only a matter of diagnosing things but also of having a clearer understanding of the semantics of what is actually happening. As a result, there are some quite explicit holes left in the type structure and the possible checking that you can do. The language is actually much stricter and more easily checked than most people suspect. In general, the compilers have been lax about doing things like subscript range checking and a variety of other things. Of course, there do remain the explicit escapes like casts of pointers.

It is possible to write interpreters that do enforce the rules of the language successfully. It turns out that if you represent pointers internally as a pointer to the data plus a lower and upper bound, then the rules are sufficient to propagate the upper and lower limits so you can do subscript checking correctly. We have an interpreter that does that, and a number of other people do too.

*To be continued.* EXE

*Many thanks to Dennis Ritchie for generously taking the time to give this interview.*

*Peter Collinson is a freelance consultant specialising in UNIX. He can be reached as* pc@hillside.co.uk *electronically (although your mailer might be happier to put the address the other way round) or by phone on 0227 761824.*

# NOVELL Develops in

# DATAFLEX

## The Benefits....

**Buyers say...**

"If I develop in DOS, I don't want to have to rewrite in OS/2" said Jim Baird, MIS manager for Novell Inc., in Provo, Utah. "DataFlex provides a good migratory path to OS/2 without losing what I have already developed in the DOS environment.

"With DataFlex under OS/2," he added, "I can put in one workstation and have a user do what would normally take two machines under DOS."

With more than 140 users accessing as many as 200,000 records and having to maintain a response time of less than one second, Baird likes the product's performance.

"I haven't had any problems with the corruption of data," he said. "I have users running 10 different OS/2 windows who can be working in a spreadsheet and come back to find the reports."

Baird likes DataFlex OS/2 as a development environment because of its macro capability for quick and simple programs and its solid programming language for more experienced users.

"You could start writing in lower-level code if you wanted to develop a quick program, and you could have a prototype out in 10 or 15 minutes," he said.

P.C. WEEK (U.S.A.)
February 19th, 1990

DataFlex provides quick applications prototyping and development through its relational database features and solid 4GL programming language. Once developed, this application can be easily run on DOS Networks and OS/2, as well as AIX, Xenix, Unix and VAX/VMS. Thus protecting the user's investment.

Novell are not alone in their use of DataFlex for applications development. Central and Local Government departments, corporates and software developers are just some of those who have been choosing DataFlex since 1981 - creating a library of thousands of successful DataFlex applications.

DataFlex is a powerful applications environment geared to on-line transaction processing and providing rapid access to data for large numbers of users. DataFlex not only protects your application development through technology portability, but also through technology developments, with the new options of a Database Engine, distributed database, Object Oriented Programming, CUA interface and relational report writer.

Try it out for yourselves with our special introductory offer of an evaluation copy with full documentation at £99 (including VAT and carriage). Send back the coupon below or phone our sales desk on **071-729 4460**.

## The performance database for applications development

*Further information from your* **DataFlex** *dealer or:*

**DataFlex (Information Management) Services Limited**
**DataNet House, 114/116 Curtain Road, London EC2A 3AH**
**Telephone: 071-729 4460**
**Telex: 94018587**
**Fax: 071-739 1247**

*All Trademarks acknowledged*

CIRCLE NO. 454

Please send me information about **DataFlex.** EV
Name ____________________
Company ____________________
Position in Company ____________________
Address ____________________
____________________ Post Code ____________________
Phone ____________________ Date ____________________
What applications areas interest you? ____________________

**DataFlex (Information Management) Services**
**114/116 Curtain Road London EC2A 3AH**
EXE 2/91

# C++ or C9X?

*When you move to C++, do you expect to be able to recompile all your old C code straight off? Derek Jones explains why you cannot, and what to do about it.*

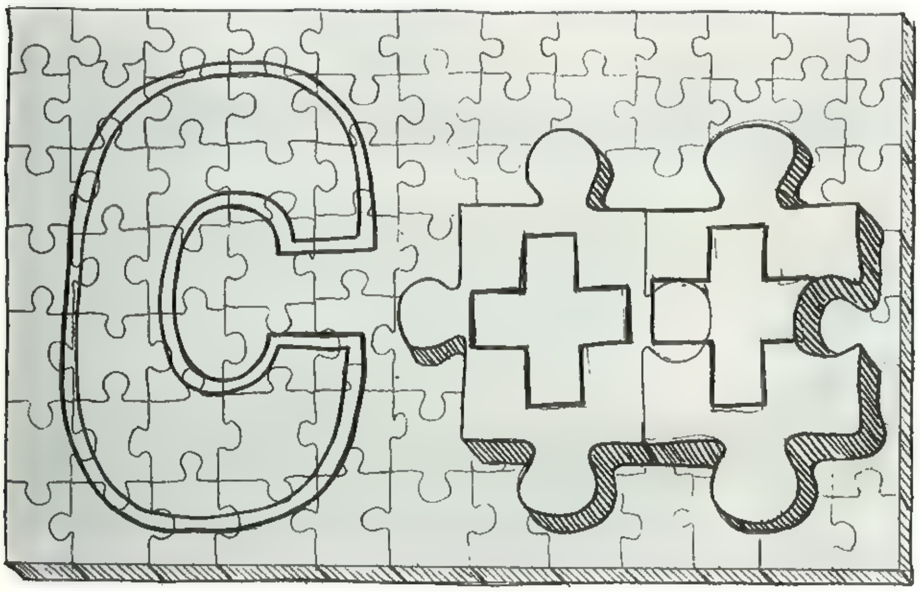
Finally, we have an ANSI standard for C. You might well assume - and it certainly seems like it - that there has been a C standard since before the outbreak of World War II. The claims made by compiler vendors in their advertising certainly give this impression. In fact, the *Draft Proposed C Standard* lost its 'Draft Proposed' prefix on 14th December 1989, only just nipping in before the end of the decade. *('The decade begins in 1991' mob please don't write in to dispute this; your views are known - Ed.)*

In this article, I do not intend to dwell on the current standard, but rather ask the question 'Where do we go from here?' The fact is that the recent emergence of C++ as a mainstream language has complicated the future of C.

C++ is widely viewed as the successor to C. Many programmers believe that once C++ has settled down (the language having just gone through a major revision), they can easily move their code from C to C++. However, things might not be that simple. Most people assume, incorrectly, that C++ is a pure superset of C.

***The C language standard has not been cast in concrete - the ANSI people are now thinking to the next revision***

This view arises partly from the fact that many C++ compilers work by translating into C as an intermediate stage. The ability to be translated to C is no guarantee of compatibility, nor is it of any particular significance. Translators are available to convert FORTRAN, COBOL, Pascal, BASIC, Ada, LISP and many other languages to C. Nobody would claim that these were supersets of C.



Thus the connection between C and C++ is not as strong as most C users probably think. Potential problems arise because the creators of C++ regarded it as a separate language, distinct from C, although they did aim for maximum compatibility with 'classic' C.

Who says that C++ is a different language from C? Three out of the four groups of people whose opinion signifies: (1) Bjarne Stroustrup, the originator of C++, (2) the ANSI C committee and (3) the majority of the current ANSI C++ committee. The fourth group is the C user community which, for the most part, is unaware of the situation.

There is a further complication. The C language standard itself has not been cast in concrete. The people who worked on the ANSI C standard, X3J11, are now thinking ahead to the next revision of the language, C9X. The standards making rules require that all language standards be regularly reviewed and updated.

So what? Well, looking ahead five to 10 years, there is the a good possibility of ending up with two C-like languages which are mutually incompatible.

The question of C/C++ compatibility is currently being debated by the C++ committee. The main purpose of this article is to widen the audience of this discussion.

## The beginning

In late 1988 ANSI (American National Standard Institute) received a proposal to set up a committee to standardise C++. Since a standards committee (X3J11) al-

ready existed for C, ANSI sent it a request asking what it thought should happen. X3J11 discussed two questions: 1) When should the C++ standardisation effort start? and 2) Should the work be done by the existing C committee?

One of the most vocal opponents against standardising C++ at once was the AT&T representative. He argued that C++ was still evolving, and that we should wait a few years before standardisation. Others pointed out that C++ was already being used for real work, and that early standardisation would reduce the number of language dialects that might have to be considered later. The voting was close, and the *Standardise now!* lobby won. The second point was more easily dealt with. The C committee regarded C++ as a different language, so the work should be done by somebody else.

(On a supernatural note, I should record that during the meeting the hotel was shaken by a minor earthquake. But as somebody pointed out, that only counted as one vote against 60 human delegates.)

Having seen the response of X3J11, ANSI set up a C++ committee (wittily named X3J16) and its first meeting was held on 15th December 1989.

## The design aims

When X3J11 started work on the C standard, it specified its aims as preserving the spirit of C and codifying common existing practice. That is: standardise what exists, and don't add anything new. To a very large degree it achieved these aims.

What are the aims of the C++ committee? Nothing has been decided yet, but the base document from which they are working is *The Annotated C++ Reference Manual* by Ellis & Stroustrup. This book includes two language features described as experimental: templates and exception handling. Various groups are looking into these areas. But the hot topic of discussion at the moment concerns those parts of C++ inherited from C that people feel should be sorted out. (These exist because the C committee's policy of codifying existing practice resulted in some of the C's quirkier knots remaining ravelled.)

## Uncle Sam's Standard

So the US went ahead and started the C++ standardisation process. This caused something of a fuss in the ISO (International Standards Organisation) which thought (and thinks) that the standardisation of C++

## Maintaining C/C++ Compatibility

The information for many of the following differences was taken from a paper by Andrew Koenig and Bjarne Stroustrup. This list is not complete.

### Identifier name spaces

C has different name spaces for object names and tag names. C++ has a single name space. Thus:

```
struct s_tag{int i,j;} s_tag;

enum count{ec1, ec2} x;

int count(void);

union u_info {int i; float f;};

typedef union u_info u_info;
```

are all legal C but illegal C++. The C programmer is relying on `s_tag` and `count` being in different name spaces. In the case of the `typedef` example, the C programmer is saving having to provide the keyword `union` when declaring an object of type `u_info`.

### Syntax

C allows redundant parenthesis to appear in declarations, C++ does not. This only affects those programmers who use parenthesis as an aid to clarifying what is meant.

```
int (*obj);

int (f)(int);
```

are both legal C but illegal C++.

### Keywords

C++ contains keywords that are not in C; `class`, `delete`, `handle`, `inline`, `new`, `operator`, `private`, `protected`, `public`, `template` and `virtual`. It is a simple matter to avoid using these reserved names as identifiers in C.

### Type safe function calls

C++ requires that functions be declared before use. C has no such requirement - but it is good C programming practice anyway. The feature that will trip up most programmers is that the C++ declaration `extern int f()` declares a function that takes no parameters, ie the equivalent to `extern int f(void)` in C. Programmers that do not explicitly declare function prototypes are in for a lot of work.

```
extern void f();

main()
{
f(1); /* Illegal C++, use differs from declaration */

g(2); /* Illegal C++, not declared */
}
```

## Maintaining C/C++ Compatibility (continued)

C++ also enforces type checking across different translation units. C has no such requirement. Thus two variables declared with different types in different files will be flagged as an error at link time. Such occurrences are anyway likely to be bugs in the C code, and ought to be flagged.

An obscure problem arises for `struct` compatibility. C++ does cross translation unit checking by type name, while C uses structural comparison. Suppose file 1 contains

```
extern struct abc{int i, j;}x;
```

and file 2 has

```
struct xyz{int i,j;}x;
```

then this is legal C but illegal C++. This is because C++ only checks the tag names, while a C linker would check the ordering and type of the fields (although the Model Implementation is the only compiler/linker known to the author that performs this checking in C). Strictly speaking, the above example results in 'undefined behaviour', but most linkers will produce the hoped-for behaviour.

### const

C++ treats objects declared with the `const` qualifier as static. C++'s `const i` is equivalent to `static const i` in C. In C, if you declare `const i` in two files, both refer to the same object. In C++, such a construct would refer to two separate objects, each local to the file in which it occurred.

To get around this potential problem, C programmers should precede declarations of `const` by `extern`, eg `extern const i` - not forgetting to give one of them an initialiser to create a defining occurrence.

### Declarations

C allows multiple definitions of the same object in the same file at global level. This is not allowed in C++. The following is legal C but illegal C++:

```
char c;
char c;
```

Such multiple declarations are usually the result of sloppy C programming, and should anyway be avoided.

### Type of 'a'

In C, character literals are of type `int`, in C++ they have type `char`. This difference is only apparent when the type is required. `sizeof('a')` returns the size of an `int` in C and the size of a `char` in C++. Character literals are still converted to `int` the same way in both languages.

### Array initialisation

C++ fixes a rather quirky C rule. In C, a string used to initialise an array of known size need not be zero terminated. So in

```
char x[2]="ab";
```

the null character terminator is omitted. C++ insists that strings be null terminated, and the above construct is illegal.

should be done as an international effort - as Bjarne Stroustrup, along with Denmark, had originally proposed. The UK faction was also very keen for this to happen.

The C++ standard is now being driven by the US. Unhappily, the UK is currently without a national voice. IST/5 (the UK committee responsible for programming language standardisation) has decided not to set up a UK C++ panel, as a protest at the US action. Since C++ is growing in popularity, and may eventually rival C as the language of choice, IST/5's current position seems short-sighted. It is very unlikely to stop the US work, and prevents the UK from influencing the direction of C++.

## What to do

Returning from the politics of Standards committees to the day-to-day problems of the programmer on the street, what can be done about the C/C++ incompatibility problem? I can see two approaches: 1) lean on the C++ committee, so that it makes C++ a pure superset of C and 2) write your code in a style that is compatible between C and C++.

Approach 2) I have described in a separate box. There are several ways of tackling Approach 1) - none offer guaranteed success. You could write to the convener of the C++ ANSI committee (address supplied at the end of the article) contributing your suggestion for the aims of the C++ standardisation committee.

***The current hot topic concerns those parts of C++ inherited from C that people feel should be sorted out***

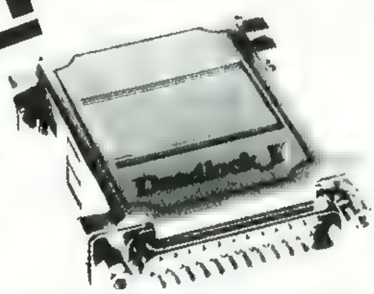
You could adopt a 'wait and see' approach. Several years hence, the committee will announce the public review of the C++ draft standard. All proposed standards must go through this period of public review. During these reviews (there are usually more than one, C went through three) copies of the draft standard may be purchased and readers may contribute their comments. These comments are collated and considered by the committee, which must give a response - although this can take the form 'Thank you for your comments, they have been considered but rejected'. The draft is updated in the light of these comments, and the cycle continues until no changes are made. However, by the time a draft standard is going through its public review, it is usually much too late to make fundamental changes.

## C9X

The C standard itself is not fixed. The rules require that standards are reviewed every five years, although in practice the cycle usually takes a decade; work does not start for five years, then takes five years to complete. So we do have a C9X to look forward to.

Work has not yet formally started on the next revision of C. But two other activities have started. The ISO has given the go-ahead for the production of an Addendum to the ISO C standard, and X3J11 has set up an interpretations committee to handle queries on the meaning of the standard.





The UK representation, among others, was unhappy with the looseness of the drafting of the C standard. This document contains many undefined features, poorly defined constructs and ambiguities. So one aim of the ISO's addendum is to tighten up the language definition - without changing what it defines. (The list of problem areas is still being compiled, so if **you** have a copy of the Standard, and you think that you have found an omission or inconsistency, please contact me at the address given below.) The addendum will also address a Danish request for an alternative, non-trigraph method of supporting characters such as [ and { in the C source, and a Japanese request for run-time support of multi byte characters.

Many of the interpretation questions raised by the UK were addressed by ANSI X3J11 at its meeting in California during September last year. The volume of the UK questions swamped the meeting and the answering of some of them had to be postponed. A surprising suggestion (in my view) was that at some future date, interpretation questions might be dealt with at ISO C meetings.

Finally, there is also interest in extending the C standard in specific areas. A numerics group has already been set up, and several meetings have been held. If you are interested in this, please contact Rex Jaeschke at the address below.

Will X3J11 eventually start work on the next version of the C standard, or will it yield to C++? We will have to wait and see.

EXE

**Some useful addresses**

**UK C++ standardisation**
Neil Martin
BSI Quality Assurance
PO Box 375
Milton Keynes MK14 6LE
email: neil@bsiqa.co.uk

**UK C Panel Convener**
Cornelia Boldyreff
Department of Computer Science
Brunel University
Uxbridge
Middlesex UB8 3PH
email: Cornelia.Boldyreff@brunel.ac.uk

**Editor ISO C Addendum**
Derek Jones
Knowledge Software Ltd
32 Cove Road
Farnborough
Hants GU14 0EN
email: derek@knosof.co.uk

**ANSI C++ Convener**
Dmitry Lenkov
California Language Laboratory
Hewlett Packard
19447 Pruneridge Avenue
Cupertino CA 95015
USA
email: dmitry%hpda@hplabs.hp.com

**Convener C Numerics group**
Rex Jaeschke
2051 Swans Neck Way
Reston VA 22091
USA
email: uunet!aussie!rex

*Derek Jones is a member of the UK C Panel and has represented the UK at ISO C meetings. He is also the technical editor of the UK component to the ISO C Addendum and the project editor of the ISO C Addendum.*

*The UK C panel meets every three months in central London. If you would like to take part, please contact Cornelia Boldyreff (address given in box) for details. Contact Neil Martin at the BSI for details of C++ standardisation.*

# Here are the Bugs - TopSpeed C

*Last year we printed a letter of appeal from the Romanian programmer Doru Turturea. JPI kindly responded by sending a copy of its TopSpeed C compiler. Here's how Doru, together with his friend Dan Somnea, repaid the kindness!*

A few months ago, by courtesy of Jensen & Partners International UK Ltd and the Editor of .EXE Magazine, we received a copy of the TopSpeed C compiler set, consisting of twelve 360 KB disks and various manuals. It so happened that we had read the *Here are the Bugs* articles by Mr Neil Martin published in .EXE Magazine October and December 1989. We decided to attempt something similar with our own compiler - not biting the hand that fed us, just giving it a friendly nip.

**We'd expected the %o specifier to take the first two digits from the "129e-2" literal**

For this paper, we tested V1.04b of the TopSpeed C compiler with V2.02 for the Integrated Development Environment (IDE).

First of all, we tried out Mr Neil Martin's examples of bugs found in Microsoft C 5.1 and Turbo C V2.0 implementations. With one exception, the TopSpeed C Compiler passed all its exams. In the case of Bug #6 presented in that paper - see below for a listing - the TopSpeed C syntax analyser did not detect the 'too large' value

```
#include  <stdio.h>
main()
{
  /* Declare test variables of */
  /* all the standard types    */

  char c;
  int  i,j,k;
  float f;
  int p;
  char s[10];

  /* Carry out sscanf tests */
  sscanf("129e-2","%c",&c);
  sscanf("129e-2","%o%d%x",&i, &j,&k);
  sscanf("129e-2","%e",&f);
  sscanf("129e-2","%p",&p);
  sscanf("129e-2","%s",&s[0]);

  /* Print results of test */
  printf("c = 5c \n",c);
  printf("i j and k = %x %x %x \n",
                                i, j, k);
  printf("i = %x \n", i);
  printf("p = %x \n", p);
  printf("string = %s \n", s);
}
```

**TEST RESULTS**

When run, the programs output was as follows:

```
c = 1                  good!
i j and k = 2 9 e      instead of 'a 9 e'
f = 1.290000           good!
i = 2                  good!
p = 129e               good!
string = 129e-2        good too!
```

*Figure 1 - The sscanf() function*

of the expression LONG_MAX * 8 - but we suppose that this is really the job of a *lint*.

```
#include <limits.h>
main()
{
  int j = 0;
  switch (j)
  {
    /* TopSpeed allows */
    /* over large value */
    case LONG_MAX * 8 :
      j = 1;
  }
}
```

The next question was: how close is the TopSpeed compiler to the ANSI C Standard? Taking Appendix A of the K&R book *The C Programming Language* (2nd edition) as our guide, we discovered the following bug. We presume that the last changes in the ANSI Standard are not relevant to this particular bug.

## The sscanf() function

**The Bug.** In Figure 1, the %o specifier seems to be treated incorrectly. We'd expected the %o specifier to take the first two digits from the "129e-2" literal. In this case, 12 octal is 10 decimal or 0A in hexadecimal code. The observed result was 2,

***The TopSpeed C package constitutes a very serious reply to other IDEs***

which is wrong. The other two specifiers %d and %x were treated correctly.

**The Fix.** Avoid the %o specifier.

## Conclusion

Despite these small problems, the TopSpeed C package constitutes a very serious reply to other IDEs. Its main trumps are:

- the editor has nine windows
- the error messages are more friendly
- you can use all the tremendous graphics primitives of Turbo C
- the MS-DOS DLL mechanism makes it closer to OS/2
- there is a powerful set of utilities subordinated to the same IDE

and so on.

We intend to continue its testing. If we discover any other relevant faults, we shall send them to you immediately!

EXE

*Mr Dan Somnea is Senior Analyst and Assistant Professor in the Cybernetics, Informatics and Statistic Department. He is a member of the Romanian Computer Science Society.*
*Mr Doru Turturea is researcher in the Institute of Computing Techniques. He is the secretary of the Romanian Computer Science Society.*
*Both the authors can be contacted at :*
*ECON Cybernetics and Informatics Dept, Bucharest, 15-17, Calea Dorobanti, Sector 1 71131, ROMANIA*

# Name Mangling in Turbo C++

*C++ achieves its type-safe linking with a technique called 'name-mangling'. Borland's Alex Lane explains how Turbo C++'s implementation works and why it differs from the cfront approach.*

Many newcomers to C++ are unaware that C++ compilers do name mangling. As they advance, however, users begin to deal with mangled names and invariably express an interest in the scheme used to encode names. This article presents the mangling scheme used in Turbo C++ and briefly touches on its advantages for C++ programming.

## Background

Although support for object-oriented programming is perhaps the most significant feature of C++ that distinguishes it from C, there are nonetheless several other ways in which C++ enhances C.

One of the features of C++ that help make it a better C is type-safe linkage which fits in with the C++ philosophy of making life easier for the programmer.

All declarations and all uses of all variables, functions and other identifiers in C++ must adhere to type rules. However, enforcing this consistency is not a job for the compiler because only part of the code is compiled at any given time. The linker, on the other hand, handles all of a program's code (at least in object code form) and is best equipped to do type checking. There are problems associated with modifying linkers, however.

First, such changes would be substantial, since linkers currently have only a limited understanding of type; second, should we actually make the modifications, users would have to deal with multiple linkers on their systems. The best course is to take advantage of the capabilities of existing linkers to deal with names embedded in object files; C++ does this by embedding the associated type as part of the name. This is where name encoding, or mangling as it's commonly called, comes into play.

## Turbo C++ name mangling

There are four basic forms of encoded names in Turbo C++:

**1. @className@functionName$args**

This encoding denotes a member function `functionName` belonging to class `className` and having arguments `args`. Class names are simply encoded directly, eg a class `className` will appear as:

@className@...

in an encoded name. The encoding of function names and arguments is discussed below.

**2. @functionName$args**

This form of encoding denotes a function `functionName` with arguments `args`.

**3. @className@dataMember**

This form of encoding denotes a static data member `dataMember` belonging to class `className`. Names of classes and data members are encoded directly eg a member `myMember` in class `myClass`:

@myClass@myMember

**4. @className@**

This name denotes a virtual table for a class `classname`. As noted previously, class names are encoded directly.

<table>
<tr><th>Character sequence</th><th>Meaning</th><th>Character sequence</th><th>Meaning</th></tr>
<tr><td>ctr</td><td>constructor</td><td>lsh</td><td><<</td></tr>
<tr><td>dtr</td><td>destructor</td><td>lss</td><td><</td></tr>
<tr><td>add</td><td>+</td><td>mod</td><td>%</td></tr>
<tr><td>adr</td><td>&</td><td>mul</td><td>*</td></tr>
<tr><td>and</td><td>&</td><td>neq</td><td>!=</td></tr>
<tr><td>arow</td><td>-></td><td>new</td><td>new</td></tr>
<tr><td>arwm</td><td>->*</td><td>not</td><td>!</td></tr>
<tr><td>asg</td><td>=</td><td>or</td><td>|</td></tr>
<tr><td>call</td><td>()</td><td>rand</td><td>&=</td></tr>
<tr><td>cmp</td><td>~</td><td>rdiv</td><td>/=</td></tr>
<tr><td>coma</td><td>,</td><td>rlsh</td><td><<=</td></tr>
<tr><td>dec</td><td>--</td><td>rmin</td><td>-=</td></tr>
<tr><td>dele</td><td>delete</td><td>rmod</td><td>%=</td></tr>
<tr><td>div</td><td>/</td><td>rmul</td><td>*=</td></tr>
<tr><td>eql</td><td>==</td><td>ror</td><td>|=</td></tr>
<tr><td>geq</td><td>>=</td><td>rplu</td><td>+=</td></tr>
<tr><td>gtr</td><td>></td><td>rrsh</td><td>>>=</td></tr>
<tr><td>inc</td><td>++</td><td>rsh</td><td>>></td></tr>
<tr><td>ind</td><td>*</td><td>rxor</td><td>^=</td></tr>
<tr><td>land</td><td>&&</td><td>sub</td><td>-</td></tr>
<tr><td>lor</td><td>||</td><td>subs</td><td>[]</td></tr>
<tr><td>leq</td><td><=</td><td>xor</td><td>^</td></tr>
</table>

*Figure 1 - Encoding for constructors, destructors and overloaded operators*

## Encoding functions

The encoded `functionName` may denote either a function name, a function such as a constructor, destructor or overloaded operator or a type conversion.

Ordinary function names are encoded directly eg:

`foo(int)` → @foo$qi

`sna::foo(void)` → @sna@foo$qv

The string '$qi' denotes the integer argument of function `foo`; '$qv' denotes no arguments in `sna::foo`. Argument encoding is explained in detail below.

Contructors, destructors and overloaded operators are encoded with a '$b' character sequence, followed by a character sequence from the table given in Figure 1. So, for example,

`operator+(int)` →@$badd$qi

`plot::plot()` → @plot@$bctr$qv

`plot::~plot()` →@plot@$bdtr$qv

The string '$qv' denotes no arguments in the `plot` constructor or destructor.

***Argument encoding is by far the most complex part of name mangling***

Encoding of type conversions is accomplished using the '$o' character sequence, followed by the distinguishing return type of the conversion as part of the function 'name'. This return type follows the rules for argument encoding, explained below. The lack of arguments in a conversion is made explicit in the mangling by adding '$qv' to the end of the encoded string. So,

`foo::operator int()` → @foo@$oi$qv

`foo::operator char *()` → @foo@$opzc$qv

The 'i' following '$o' in the first example denotes an `int`; the 'pzc' in the second example denotes '`near` pointer to an `unsigned char`.

## Encoding of arguments

The number and variety of combinations of function arguments make argument encoding by far the most complex part of name mangling.

Argument lists for functions begin with the character sequence '$q'. Type qualifiers are then encoded as shown in the following table given in Figure 2. Encoding of built-in types follows that for applicable type qualifiers, in accordance with Figure 3. Finally, encoding of non-built-in types follows that for applicable type qualifiers (See Figure 4).

The appearance of one or more digits indicates that an enumeration or class name follows; the value of the digit(s) denotes the length of the name. For example:

`foo::myfunc (myClass near&)` → @foo@myfunc$qr7myClass

`foo::myfunc (anotherClass near&)` → @foo@myfunc$qr12anotherClass

A character 'x' or 'w' may appear after 'p', 'r', 'm' or 'n' to denote a `constant` or `volatile` type qualifier, respectively. The character 'q' appearing after one of these characters denotes a function (whose arguments follow in the encoded name, up to the appearance of a '$' character, which is in turn followed by a return type). So,

`foo::myfunc (const char near*)` → @foo@myfunc$qpxzc

`func1(const int)` → @func1$qxi

`foo:myfunc (int (near*)(int,int))` → @foo@myfunc$qpqii$i

Array types are encoded as 'a', followed by a dimension encoded as an ASCII decimal number and a '$', finally followed by the element type:

`foo( int (*x)[20] )` → @foo$qpa20$i

Encoded arguments are concatenated together in the order of appearance in the function call. Where a number of identical non-built-in types are function arguments, the character 't' followed by an ASCII character is used for encoding the arguments. The ASCII character denotes which argument type to duplicate, and ranges from '1' to '9' and 'a' onward.

```
post::func1
   (double,double,double,
   int,int,int,
   long,long,long,
   char near*, char near*,
   char near*) →
```
@plot@func1$qdddiiilllpzctata

The duplicate 'ta' character sequences at the end of the encoded name denote the tenth argument, encoded as 'pzc'.

## Why Turbo C++ is better

The encoding scheme used by Turbo C++ represents an improvement over the scheme used in AT&T's *cfront* compiler. The cfront scheme was outlined in a paper by Bjarne Stroustrup titled *Type-safe Linkage for C++*, and was largely incorporated

| Character sequence | Meaning |
|---|---|
| up | huge |
| ur | _seg |
| u | unsigned |
| z | signed |
| x | const |
| w | volatile |

*Figure 2 - Encoding of type qualifiers*

| Character sequence | Meaning |
|---|---|
| v | void |
| c | char |
| s | short |
| i | int |
| l | long |
| f | float |
| d | double |
| g | long double |
| e | ... |

*Figure 3 - Encoding of built-in types*

| Character sequence | Meaning |
|---|---|
| <a digit> | (an enumeration or class name) |
| p | near * |
| r | near & |
| m | far & |
| n | far * |
| a | array |

*Figure 4 - Encoding of non built-in types*

into *The Annotated C++ Reference Manual* by Ellis and Stroustrup. Far from claiming its scheme to be a 'standard', the latter work notes that it would be unwise to use identical encodings of type signatures in systems that are not link compatible, which currently is the case among C++ compilers for DOS. There are several ways in which the Turbo C++ encoding scheme improves on the cfront scheme.

First, the cfront encoding scheme was designed with C as an intermediate language. Using this scheme, it is easy to look at an intermediate C file and determine whether a name is encoded, and to decipher any additional information. This feature is of limited use, however, when using a native-code compiler such as Turbo C++ since there is no intermediate C code and the need to view mangled names occurs only rarely.

Second, the cfront scheme is intended for systems that have a single type of function call and a single pointer type. C++ compilers for DOS must deal with the keywords `near` and `far` used in function and pointer declarations, and so must depart from the cfront scheme to be workable.

Third, using the cfront encoding, it is possible (though improbable) to create names in C that, when encoded, look like C++ mangled names. For example, the C variable declaration

***Using the cfront encoding, it is possible to create C names that look like C++ mangled names***

```
int foo__Fi;
```

would be encoded as 'foo Fi', which corresponds to the C++ encoding for the declaration

```
int foo(int);
```

which will offer problems later at link time. This problem will not occur using the Turbo C++ encoding scheme.

Finally, a scheme like the one used in Turbo C++ makes it easier (and faster) for associated tools such as debuggers to find and process mangled C++ names. This ability to support tools is of particular importance for the future of C++ and object-oriented programming.

## Conclusion

In this article, we've taken a detailed look at the Turbo C++ name mangling scheme. The mangling scheme used by Turbo C++ takes advantage of the native-code nature of the compiler and the MS-DOS environment in which it operates, and expedites the use of tools in the development of C++ programs.

EXE

*Alex Lane is Senior Technical Writer for the Languages Business Unit at Borland International in California. He is also a regular contributor to the monthly newsletter The C++ Report.*

# Inside the Indexes

*Comparisons of dBASE products tend to concentrate on such points as the programming language commands and data presentation. Ken Sawyer gets down to the heart of the matter: the design of the index files.*

Most database products use a B+ Tree for maintaining database information in sorted orders. dBASE, Clipper and FoxPro are no exceptions. However, interesting choices can be made on the implementation details.

The most critical choice is the block size. This is the number of bytes read from disk or written to disk when manipulating the B+ Tree. The whole point of using a B+ Tree (rather than a much simpler binary tree) is that it minimises the number of times the disk is accessed. This is important because, even with the latest drives, accessing disk storage takes thousands of times longer than accessing RAM memory. As the block size becomes larger, it is necessary to access the disk less. However, there is a trade-off because large block sizes can require more RAM memory and it can take longer for any given disk access.

Regardless, the most interesting point is that dBASE III NDX index files are slower than Clipper NTX files and dBASE IV MDX index files. This is because the block size of dBASE III NDX files is fixed at 512 bytes, the block size of dBASE IV MDX files defaults to 1024 bytes, and the block size of Clipper NTX files is fixed at 1024 bytes.

The choice of 512 bytes for a block size under MS-DOS is particularly bad. This is because MS-DOS always writes 1024 bytes at a time on 1024 byte boundaries. Consequently, if you tell MS-DOS to write bytes 0 through 511 to disk, MS-DOS will read bytes 0 through 1023 from disk, modify bytes 0 through 511 and then write bytes 0 through 1023 to disk. This means that it would have been much faster to tell MS-DOS to write bytes 0 through 1023 than to tell MS-DOS to write bytes 0 through 512. Similarly, if you tell MS-DOS to read bytes 0 through 511 from disk, MS-DOS will read bytes 0 through 1023 and then give you bytes 0 through 511.

**Clipper's design adds unnecessary complexity, forces a greater code size, wastes disk space and valuable RAM**

Note that under OS/2 Version 1.0 the magic number changes from 1024 to 2048. This means that under OS/2 Version 1.0, 2048 would be a better block size than 1024.

There are some other significant differences in the internal index file formats. Within an index file block, Clipper NTX files implement another level of indirection. Rather than just having a list of sorted keys, there is a list of random ordered keys followed by a list of integer pointers to the keys. These integer pointers specify the key ordering. The idea is that it is faster to move and manipulate the integer pointers within memory than the simpler approach of moving the keys themselves.

My intuition says that this was a poor design decision. This design adds unnecessary complexity, forces a greater code size, wastes disk space and wastes valuable RAM. Spare memory can be used to increase performance in many different ways. One of the best ways is to buffer disk information to reduce disk I/O.

## A date with Clipper

Another thing that Clipper does is represent keys entirely as ASCII. For example, under dBASE, a date key is saved as a floating point double representing a Julian day. Under Clipper, a date key is held as eight characters. The date July 4th, 1990 would be '19900704' - the same way a date is stored in the dBASE DBF data file. The Clipper method is superior because no translations from the data files are necessary, and because it is quicker to compare eight ASCII characters than a floating point double.

Finally, many people assume that the FoxPro IDX file format is superior to dBASE and Clipper because FoxPro has a reputation for speed. However, FoxPro has good performance because of its internal memory management. It is optimised for single user performance. In a single user environment, it is possible to work entirely in RAM memory as long as there is enough memory available. Unfortunately, this is not possible in a multi-user environment on a network. In a multi-user environment, it is necessary to keep reading information from disk because it gets changed by other users.

For best performance when using a B+ Tree, the implementation details are critical. These details include disk file formats as well as internal memory management.

EXE

*Ken Sawyer has participated in the development of Code Base 4.2, Sequiter's C library which implements complete dBASE and Clipper functionality. You can contact Sequiter Software Inc on 0101 403 448 0313.*

# Now we are Five

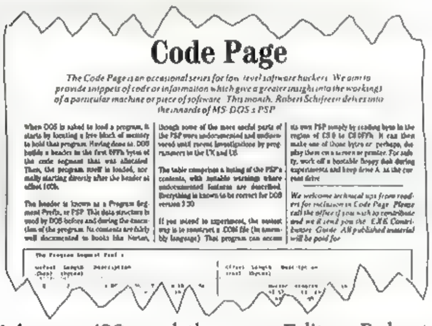
*This Magazine is five years old this month. We look back at some of the highlights (or otherwise) of its existence.*


The first issue of .EXE is published in February 1986. The cover feature is an introductory article to Intel's new 80386. There are also pieces on developing for DR's GEM, a survey of compilers costing less than £100, and a general feature on windowing environments which compares GEM, Microsoft's Windows and Epson's TAXI. *(Wot that? Ed)* Ian Adams' editorial promises '... we provide a forum for you to exchange views and problems.' The magazine contains no program code.

## Cross Development on VAX

*Roger Farrington believes embedded systems programmers should be using cross-assemblers and compilers on host systems.*

Issue 5 is in October '86. As a result of a dispute with its mailing house, the magazine has been 'off the air' since June. It is still thinking big, however, as this VAX article shows. This issue marks the debut of our current production manager, who styles herself 'Katey Matey' on the masthead.



## Code Page

*The Code Page is an occasional series for low level software hackers. We aim to provide snippets of code or information which give a greater insight into the workings of a particular machine or piece of software. This month, Robert Schifreen delves into the innards of MS-DOS's PSP*

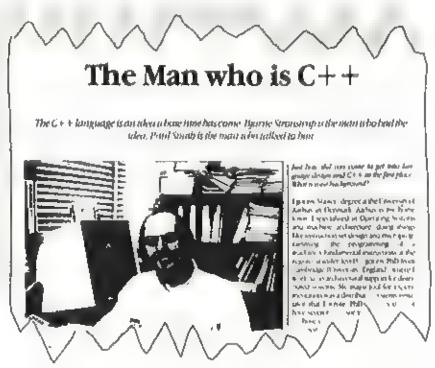
February '88, and the new Editor, Robert Schifreen, is a tireless innovator and supporter of the IBM PC. This, the first article in the Code Page series, covers the undocumented features of the MS-DOS PSP.

## Regina v. Schifreen & Gold

In April '88, Robert Schifreen and accomplice Steve Gold succeed in their House of Lords appeal against a conviction for hacking Prince Philip's Prestel account. Understandably chuffed, Schifreen publishes the full text of the judgement in June.


A biscuit introduces MS-DOS V4.0 to the world, and a new columnist to .EXE's readership: Ms Verity Stob. In her October '88 piece, she offers a spoof quiz to determine the user-friendliness of your software. Also in this issue we review the second edition of the C programming language by Kernighan and Ritchie - needless to say we like it! The magazine is now very PC-oriented.

Paul Smith's interview with Bjarne Stroustrup (which, despite its spontaneous feel, is mostly conducted via email) in December



## The Man who is C++

'89 is a great success, stimulating renewed interest in both C++ and .EXE itself. For the first time since 1986, cartoons have begun to appear in .EXE's pages - notably a Glen Baxter imitation portraying a man with a monster bug in his program. A real time supplement includes an article describing London's new Docklands Light-Railway.

In June '90, Timothy Wegner of the extraordinary Stone Soup Group explains how to plot beautiful 3D fractals. We offer a free copy of his software to anybody who sends in a disk. Nearly everybody does.

July '90, and a new Editor brings a UNIX columnist, a Soapbox for the opinionated, a range of thematic issues covering such subjects as computer security and Pascal, and a bizarre taste in front covers. Jules May didn't use C, provoking some combative correspondence from those who did. Our July cover featured a controversially intestinal view of Windows 3, so to avoid re-opening old wounds, here is the Chris Duggan's near-universally popular Ratty, Mole and Mr Toad ('Oop-oop!') from November.


EXE

# A new Smalltalk

*Steve Cook has a look at Objectworks\Smalltalk Release 4, the new Smalltalk from ParcPlace Systems, now integrated with native window managers.*

Smalltalk originally dates from research done at the Xerox Palo Alto Research Center in the early 1970s. At PARC, Smalltalk was the programming environment for Alan Kay's vision of personal computing - the 'Dynabook', a small networked handheld personal computer with a high-resolution display. Smalltalk first became widely known in August 1981, when BYTE magazine featured a collection of articles about the system. Over the next couple of years, commercial implementations of the version called Smalltalk-80 became available from Tektronix, Apple and ParcPlace Systems. Tektronix and Apple subsequently dropped out of the Smalltalk business, while ParcPlace Systems (which spun out of Xerox specifically to develop Smalltalk) has continued to produce versions of Smalltalk-80, the most recent being version 2.5.

Smalltalk is not easy to learn. Apart from its sheer size, it demands a shift of perspective which can be difficult to grasp for people used to more conventional programming technology. It is quite usual to spend several months getting familiar with the system; becoming a real expert can take years. Up to now, the most effective use of Smalltalk has been in research laboratories, prototyping novel ideas (Randall Smith's Alternate Reality Kit is a vivid example).

Recently, with the appearance of window managers on host platforms, the Smalltalk-80 policy of taking over the entire machine and managing its own windows has become less and less appropriate. The original window manager was written under the assumption that the only window allowed to change is the active one. Windows updated by asynchronous processes, such as clocks or incoming data from a network, would simply splatter updates through any windows overlapping them. This limitation persists in Version 2.5 and, together with the 'flap-out' scrollbars and absence of colour, makes it appear eccentric and obsolete in a modern PC or workstation environment. Digitalk's Smalltalk/V, a less sophisticated system targeted at the PC and Mac marketplace, is better integrated with host window systems, and provides an attractive and cheaper alternative.

***Objectworks\Smalltalk Release 4 is not for creating shrink-wrapped applications***

Now comes Objectworks\Smalltalk Release 4, the new Release from ParcPlace, available now on Apple Macintosh and on several UNIX systems under X, soon to be Released under Microsoft Windows 3.0. Objectworks\Smalltalk Release 4 discards the old Smalltalk-80 window manager and instead uses the platform's native window manager for handling its windows.

I have been programming in Smalltalk-80 for a total of seven years, and have built some fairly large systems in it. My burning question, when I got my hands on Objectworks\Smalltalk Release 4, was whether at long last this is a Smalltalk-80 system for building portable, fully-integrated workstation applications as well as research prototypes. The answer is yes, but with some strong reservations. This preview is based on my experiences with Objectworks on an 8 MB Macintosh IIx.

## Smalltalk programming

Most of Release 4 is very familiar. The Smalltalk language is almost unchanged, except the assignment symbol is now the familiar := (in the previous version a left arrow symbol could also be used). Smalltalk programs are structured in terms of 'sending messages to objects'. Each message, when it arrives, causes the execution of a corresponding 'method' defined in the object's class. The dynamic association of methods with messages, often called 'dynamic binding', is the feature which gives Objectworks\Smalltalk much of its power.

Programming in Objectworks is not like programming in C or Pascal. There is no main program, no apparent compilation and no linking. Instead, the programmer alters an Objectworks image incrementally, changing existing methods and introducing new ones. At any point the image can be saved in a file. It can subsequently be restarted, recreating the same situation as when it was saved, including window placement and (with some exceptions) window contents.

Although Objectworks feels just like an interpreted system, it actually compiles methods into machine code on the fly. This means that an Objectworks image developed on a 386 PC can be run on a 68020 Macintosh without alteration. When running on the PC, the run-time system compiles methods into 386 machine code; on the Mac, 68020 code will be produced. All of this is completely transparent to the programmer.

The original Smalltalk was designed to be used by a single person. Smalltalk-80 provided tools for managing projects and changes, but these are only usable for groups of two or three programmers. For projects of larger size, additional third party tools are necessary. There are a few companies (such as Instantiations Inc. of Portland, Oregon) making such tools for Smalltalk-80, and with luck these will be updated to work with Release 4.

Apart from the different look of the windows, and a few additional options here and there, the programming tools in Release 4 are just as they were in the earlier system. The system browser is used for navigating around the class hierarchy, reading, creating and editing code. The workspace is used for saving and evaluating code fragments. Any object can be inspected, with a window giving access to its state. The debugger was always one of the best around, and remains so. The file list, file editor, system transcript and change list browser complete the repertoire of tools.

Earlier versions of Smalltalk used a pop-up menu on the screen background to launch new windows and utilities. In the native windowing environment, Smalltalk has no access to the screen background, and this menu is replaced by a window called the *launcher*. The trouble with the launcher as supplied is that it is easy to lose under mounds of windows. To fix this, I spent a few minutes adding a menu item to each open window, allowing me to 'unbury' it. There are not many systems around which are open enough to allow changes to be made so easily at such a fundamental level, and this strength of Smalltalk is preserved in Release 4.

## Digging deeper

Digging into the code revealed substantial differences between Release 4 and the earlier systems. Some familiar classes, like `Form` and `BitBlt`, have vanished completely, to be replaced by `Image`, `Pixmap` and `RasterOp`. The Form Editor and Bit Editor have gone too. There is no longer a global called `Display`; in Release 4 everything must be done via a host window. There are many new classes supporting colour, and the user interface classes are very different.

On the other hand, many parts of the system remain the same. `Numbers`, `Collections` and most of the `Kernel` and `System` class categories are unchanged. `Exceptions` and `Processes` are just as they were. It took me just a few hours to understand how Release 4 differed from its predecessors, and although the differences are substantial, the overall feel of the system is extremely similar.

Smalltalk's user interface is structured using a principle called Model-View-Controller (MVC). The basic idea is that the Model is the data to be interacted with, the View is the rendering of that data on the screen, and the Controller provides the mechanisms for mapping input events into changes to the Model. Release 4 retains the MVC principles, but the view hierarchies and protocols are quite different. Reflecting the fact that some objects exist in the host window server rather than in the Objectworks image itself, Release 4 has `Windows` as well as `Views`. Each window on the screen is represented by an instance of the class `ScheduledWindow`. The parts of windows are represented by classes such as

***The debugger was always one of the best around, and remains so***

`VisualComponent` and `CompositePart`. Each part of a window is held in a `Wrapper` object, which adds borders and controls, as well as doing translation and clipping. For some reason, the hierarchy of visual classes has been designed so that only non-composite views can scroll. This seems an unnecessary restriction.

In the window management scheme of Release 4, the host window manager does not guarantee to retain the graphical information written onto its windows. It is up to the programmer to provide a method which can fill in the parts of windows exposed by window operations. This is often called 'damage repair'.

## Colour

Objectworks\Smalltalk Release 4 introduces full colour graphics. The whole graphics subsystem is completely different from earlier versions, and is dignified with a new name: the Smalltalk Portable Imaging Model (SPIM). Displaying graphics in SPIM is accomplished through graphics contexts associated with graphics media. A graphics context specifies various parameters, including default paint colour, font, and line display attributes. Graphical objects are displayed either by sending messages direct to the graphics context, or by sending messages to graphical objects with the graphics context as a parameter. The result is displayed using a stencil and paint principle, similar to that used in Postscript. Texts, polylines, polygons, arcs, wedges and images are supported.

All co-ordinate systems in Release 4 are pixel-based with no facilities for scaling. This is a step back from earlier Releases which supported both scaled and translated co-ordinate systems.

Some effort has been taken to ensure that colours in Release 4 are portable between platforms. Colours can be specified by red-green-blue (RGB) or by hue-saturation-brightness (HSB) co-ordinates. Pixel images up to 24-bits deep are stored in a portable format, and will appear with the right colours on any machine that Objectworks\Smalltalk runs on. If the colours specified cannot be rendered on the machine, Release 4 will do the best it can, including dithering (a technique which gives adjacent pixels different colours, relying on the eye to average between them).

The previous Smalltalk font facilities were underwhelming. Font technology has developed at an extraordinary rate over the last few years. Release 4 provides a nifty mechanism whereby you can issue a font query to the underlying system, and it will do its best to match what you asked for. If you want a specific font, you can say so and the system will give you it if it can; otherwise you can specify various attributes (like whether or not you want a serif font, and how big and bold you want it) and it will find you something near.

Not included with the basic system, but as a set of 'fileIns' - source code files which you can easily compile into the image using the file list tool - is support for Postscript printing. I did not have a Postscript printer available for this review, so I couldn't try these features out, but from the code they looked fairly comprehensive.

## The user interface

Running Release 4 and Release 2.5 side-by-side really shows up how much of an improvement there is. Release 4 looks thoroughly modern in comparison. By judiciously configuring a set of defaults in a class called `LookPreferences`, you can get nice coloured backgrounds and foregrounds, with a pleasing 3D effect. Conspicuous by its absence is a `LookPreferences` browser tool, which is now high on my list of 'goodies' to write or find. (Goodies is a term used in the Smalltalk community to describe small public-domain accessories for filling in. These can often be found on various networks.)

The scrollbars are more aesthetically pleasing than the old flap-out ones. They come narrow and a little fiddly, but the width can be adjusted with `LookPreferences`. A range of buttons is provided, including radio buttons, checkboxes, press buttons etc. These do not have a 3D press effect. I was disappointed to find that the attributes of buttons (text font, size, colour etc) are kept in a system global; it would make more

sense for them to be associated with each individual button.

Each pane in a window has a menu bar along the top, and buttoning within this bar brings up the operate menu (corresponding to the middle button with a three-button mouse). This is useful on the Macintosh, because of the one-button mouse; on a system with a three-button mouse, these menu bars are irrelevant. On the Macintosh I would like another area to button in, bringing up the window menu - or better, have the window menu on the normal Mac menu bar. Having to press the command key and click to get at the window menu is not ergonomic.

I was hoping the system was going to be able to adopt the look and feel of the platform it is running on. Ideally, I want to develop an application on the Mac, using Mac scrollbars, dialogs etc, and then just lift it and drop it onto Motif or Windows 3.0 and have it automatically adopt the local user interface components. Release 4 does not do this. It simply uses the host window system to provide it with rectangles to draw in; within these rectangles it looks the same regardless of which system it runs on. This means that an application created under Objectworks will not look like a normal Mac or Windows application. On the Mac in particular, there are only two menus in the menu bar. One is a file menu, which only allows the user to quit the application and, for some reason, to bury the current window. The other is an edit menu, but this was greyed out and I could not find any way of making it active. Cut and paste does not work between Objectworks windows and Mac windows. The scrollbars do not look like Mac scrollbars. Particularly annoying in the Mac context is the absence of the normal resize box in the bottom right-hand corner of each window: instead, you have to command-click in a window and select resize from the resulting pop-up menu.

I found a fileIn which contained some code for creating Mac menus and dialogs, and I managed to make it work, so it is certainly possible to use these underlying facilities; but the code is not portable across platforms. A further fileIn allows PICT files to be imported from the clipboard, and I used this successfully to import an image from MacPaint and display it within Release 4. There is also a fileIn for accessing the Mac's sound resources.

A normal Macintosh application uses entities called resources to implement its windows, menus and dialogs, and it is a pity that Objectworks\Smalltalk provides no interface to these. A programmer wishing to use them would have to start from scratch, by linking his own primitives into the Objectworks run-time system.

***Smalltalk is not easy to learn - it demands a shift in perspective which can be hard to grasp***

There is a trade-off between portability and the ability to support the detailed features of each platform. Objectworks\Smalltalk has focused on portability, with great success - you really can take an image developed on one platform and run it on another. It would be nice if there were additional facilities to access the underlying features of each specific host platform, and it would be even nicer if these were organised so that an application would adopt the look and feel of the platform it is on. Still, there is nothing in Objectworks\Smalltalk that will stop you doing these things for yourself if you are brave and have the time.

## Memory

Smalltalk has always been strong on memory management. The programmer doesn't need to worry about memory, as it is all managed automatically. There is no way a pointer in Smalltalk can refer to anything except a valid object, and when there are no more pointers to an object its space is reclaimed. Release 2.5 was pretty sophisticated at managing its memory, and Release 4 is even more so. Memory is divided into different zones which are handled according to what kinds of objects live there. Short-lived objects are collected using a technique called scavenging, whereas longer-lived objects are collected incrementally. In Release 4, the incremental garbage collector runs as a background process within the Smalltalk image itself, and various parameters of the memory management policy can be altered by the programmer.

An interesting new feature in Release 4 is the class `WeakArray`, implementing 'weak pointers'. These are pointers which do not inhibit garbage collection. They can be set up to detect when objects die, and run finalisation code. They can be particularly useful for distributed algorithms, running in several communicating images. For example, a distributed garbage collector might want to inform other images whenever an object dies, to update global reference counts.

During several hours of normal program development with Release 4, I didn't notice the system garbage collecting or compacting at all. I did manage to make it work hard by writing some code that repeatedly allocated large pixmaps; this caused the system to stop for a second or so at intervals to garbage collect and compact memory. The system does use a lot of memory. My experience with the earlier version suggests that 8 MB is a practical minimum for writing systems of any significant size.

## Performance

On the Mac IIx, my (beta) copy of Objectworks was reasonably responsive, although there was a perceptible delay between pressing the mouse and getting the result. In comparison with some other Mac applications, such as MacWrite, Objectworks was noticeably sluggish, although to be fair I didn't have any full colour applications with which to compare it. Sometimes, for no apparent reason, the system didn't respond and I needed to try twice to get a selection or menu. There were no crashes while I was testing it.

## Documentation

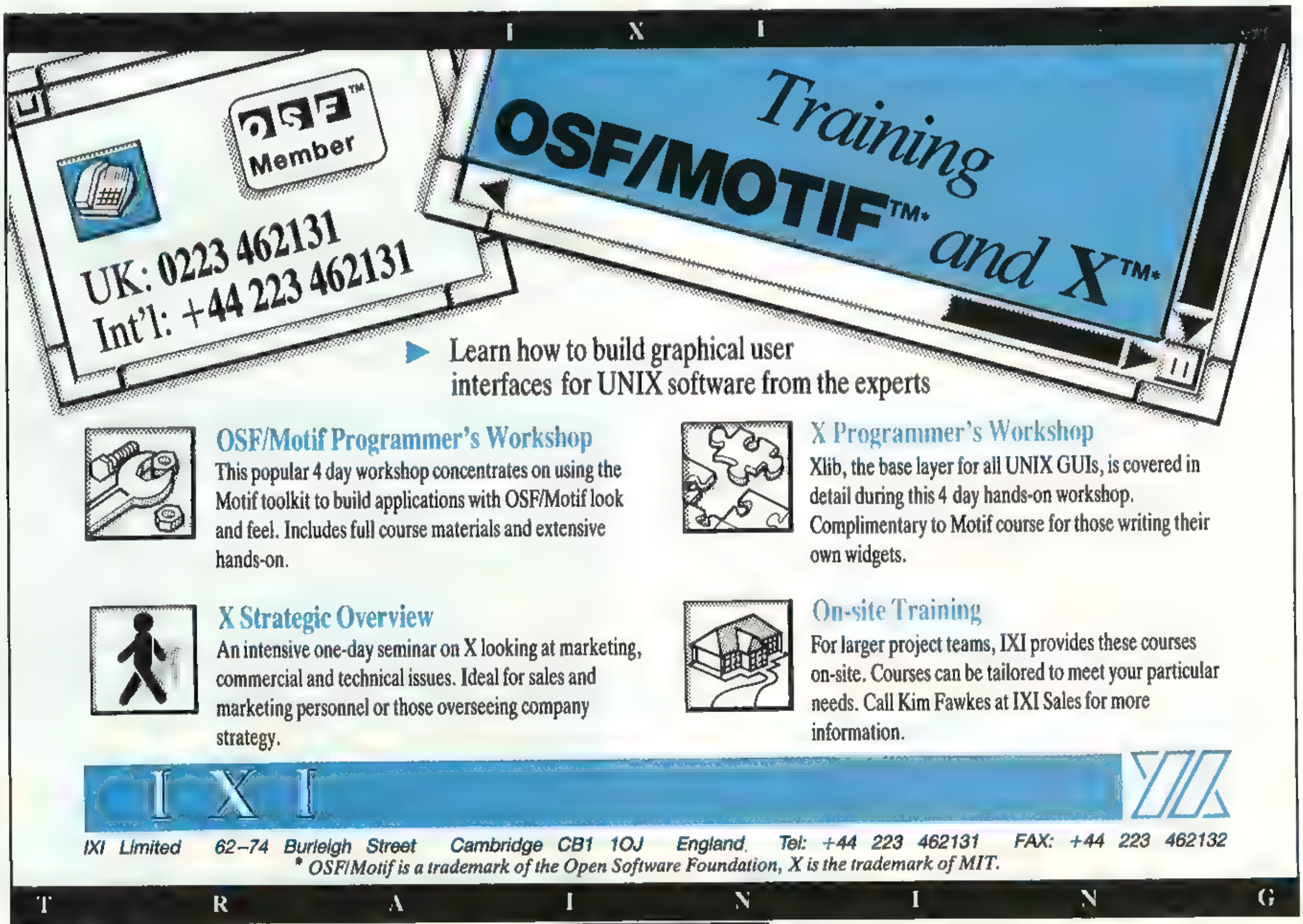
The 300-page manual is comprehensive, and covers most aspects of the system in adequate detail. It includes a section on how to port existing Release 2.5 applications to Release 4, grading the impact of the system changes and the difficulty of conversion for various types of application. Applications which use graphics extensively are greatly impacted and are difficult to port, whereas applications using only text are relatively easy.

Much of the documentation is contained within the code itself. I found some of the comments in Release 4 to be incomprehensible, and several to be just plain wrong. For a system as difficult to learn as Objectworks, it can be discouraging to come across incorrect comments, and this is one area where I think a lot of improvement could come from a relatively small amount of effort.

## What's it for?

Objectworks\Smalltalk Release 4 is not for creating shrink-wrapped applications. Without a lot of extra work, developments with


CIRCLE NO. 475




CIRCLE NO. 512

Objectworks will have their own individual look and feel, and will not integrate readily into the native environment. The product comes into its own when portability across platforms is more important than integration with the native environment. In this case, Objectworks is probably the most portable system around.

Traditionally, important application areas for Smalltalk have been in rapid prototyping, and in data analysis and modelling in large business applications. Objectworks\Smalltalk Release 4 is a substantial improvement on earlier versions, and makes Smalltalk an even more attractive proposition in these areas where it is already strong.

*Figure 1 - Objectworks\Smalltalk Release 4 screen display on Mac*

The next step would be real integration with host systems, combined with binary portability, and these improvements would open up new markets for the product.

Over the years, Smalltalk has been one of the most influential pieces of software in existence. Smalltalk gave the world windows, mice, pop-up menus, mode-free interaction and object-oriented programming. It is curious that such an influential system should now be catching up with its own legacy.

EXE

*Steve Cook is Managing Director of Object Designers Ltd, specialists in the introduction and application of object-oriented technologies. He has used Smalltalk-80 since 1983. He can be reached on 0279 755396.*

*Objectworks\Smalltalk Release 4 can be obtained from AI International (0923 247707), price £2950 for all platforms. It is available now for Apple Macintosh, Sun SPARCstations and Digital DECstations. It will be available in Q1 1991 for IBM RS/6000, HP 9000 Series 300, Apollo workstations and MS-DOS with Windows 3.0.*

CIRCLE NO 481

# No excuse for user abuse

*Have you ever called in an HCI consultant?*
*John Harris explains what he does and why he is needed.*

I've just returned from INTERACT '90, a three-yearly international conference on HCI (human-computer interaction). A speaker, who used a PC and LCD transparency on the overhead projector instead of conventional slides, had us laughing at some classic design misconceptions.

When he'd finished with his first screen, he read out the last line *Hit any key to continue.* Marvelling at the choice before him, he slowly put his left hand to his mouth in exaggerated thought, as his right index figure hovered over the keyboard, finally pressing the top left-hand key. It was, of course, **Esc**ape... and the screen went blank. The audience erupted in laughter at such a graphic demonstration of a well-known design *faux pas.*

It's funny to see old chestnuts acted out so meticulously, but tragically they still occur in too many systems, wasting people's time, confusing them and causing them to make errors.

In this article I explore some of the things that go wrong and start to unravel what lies behind them. At least three communities are involved; developers, users and HCI experts. Although HCI experts may seem the most relevant, little can change unless developers and users play their part. I'll look at each in turn.

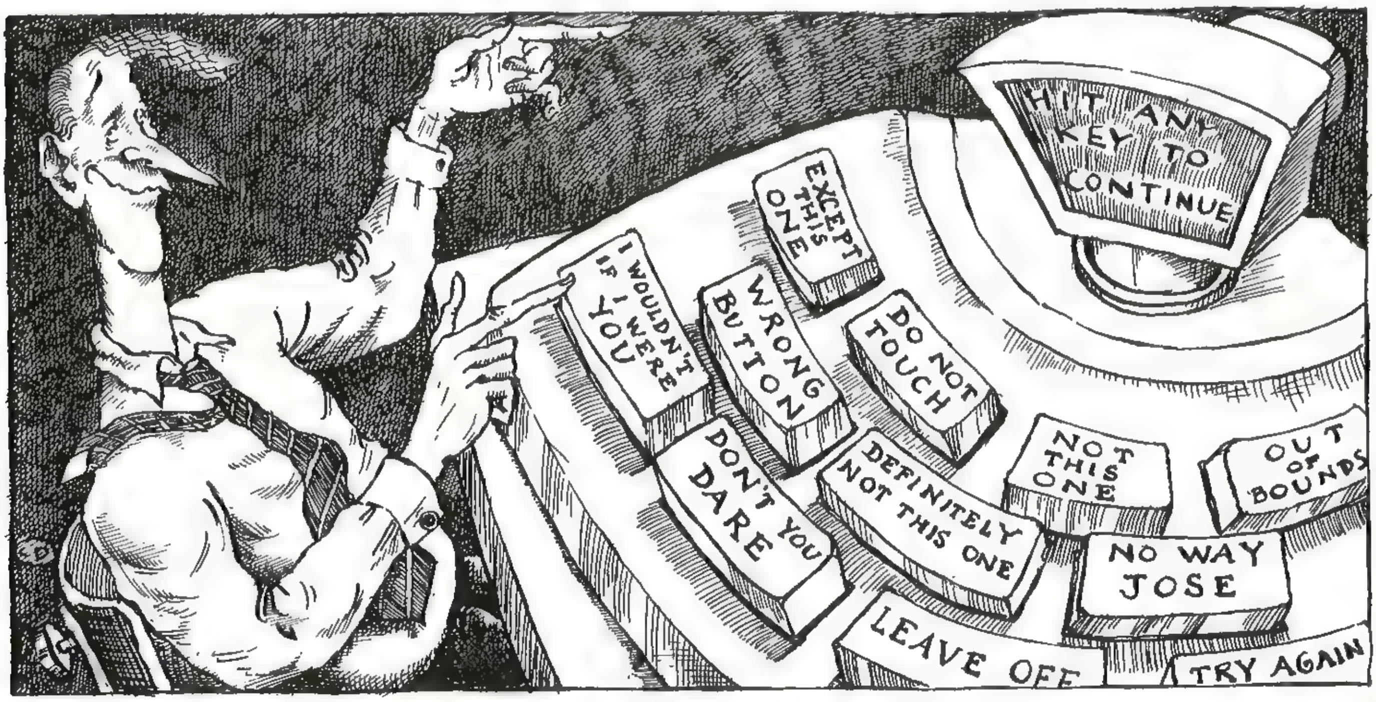
## Software developers

Developers tend to work with a systems approach. Priority is given to data and functions, and not to the user's-eye view, or how users' tasks are organised. This isn't surprising - computer professionals are seldom trained in human factors.

This 'computer-oriented' view shows up in the approach. In the early days, models of the development life-cycle were linear, and not amenable to techniques such as prototyping. Once a stage had been completed and signed off, you never went back to it. More recently, structured methods and development models have become more tractable, allowing some degree of iterative design. But the malign influence of the past is still there.

Sometimes user interfaces are grafted on once the 'important' parts of the program are complete. The resulting problems sometimes lead to HCI consultants, like myself, being called in towards the end of a project. I have to tread carefully. Imagine the situation: you're on the point of completing a project, when an outside consultant is brought in to tell you how you've got it wrong!

There may be good reasons why things have been done the way they have, and I'm careful to get the confidence of the development staff. Equally they seek my confidence by showing they know about user interfaces. Sometimes they're amazingly well-informed. On other occasions, they



reveal that they're blindly following a guideline or quoting a saying, and they don't understand the issues.

## Mythology trap

When guidelines are part of a thorough method they can be helpful. In designing user interfaces, however, mythology is as prevalent as methodology. It's a mythology that often lets designers down, and it's worth examining some examples.

Although I call it a mythology, its individual components - rules, guidelines and sayings - often contain more than a grain of truth. But as they get passed around out of context, becoming part of design folklore, they lose their *ifs* and *buts*, and get so distorted they finish up more of a hindrance than a help. A good name for such misleading guidelines would be *mythguidelines*, but I'll settle for misguidelines. Let's examine some misguidelines about menus, a very rich source of them.

A good example is *menus are user friendly*. It crops up in various forms and places; in advertisements and sales literature for menu-driven software, and is quoted by people wanting to prove their awareness of human factors.

The reason it's misleading is because it begs important questions; blind application just results in a mess. The begged questions are about the type of user (new, occasional, frequent, typist, non-typist and so on), the type of menu, application, host system and so on.

The myth has sprung from the genuine observation that menus make options and responses explicit, lessening the need to look up command names, syntax and availability. However, in some forms it carries the implication that a system must be user-friendly just because it's menu driven, or that menus are always preferable to alternatives.

Often there's no single best interactive style; strengths and weaknesses may balance, and more than one adequate solution may exist. If menus are chosen, you still need to implement them with care.

Menus can grow into enormous hierarchical monstrosities on complex systems. This is most apparent when they're implemented as one-menu-per-screen, rather than pull-down or pop-up ones. As more and more functionality and/or data classes are added, the hierarchy grows and grows. To use a particular function or data set, you may have to go via several menus, only to come all the way back immediately afterwards.

I met an extreme example in a financial dealing system. Users' actions changed the data it held, such as current prices, or who owned what. The user interface made it more like editing a database than dealing: simple deals involved convoluted sequences of actions, as you scaled up and down menus fetching records to edit and functions to edit them with.

***A good example of a classic myth is 'menus are user friendly'***

Designers can provide short cuts, producing a network of menus rather than a hierarchy, but this adds to the complexity - it becomes more difficult to learn the system and easier to get lost in it. In most cases, it's better to structure the interface around the tasks the user wants to perform on the system rather than the functions or data available on the computer. Once the user has selected a particular task, the computer should make easily available the functions and data needed to perform it.

## Seven Up

You can cut down the number of menus by having more options per menu. This is the subject of another misguideline, *don't have more than about seven items in a single menu* (the magic number quoted varies). The genuine origin of such a restriction is the limited capacity of human short-term memory. When you're making new and difficult choices you may need to hold the options in short-term memory.

However, experienced users may know the different options (long-term memory has enormous capacity) and prefer to have them all presented simultaneously, rather than picking their way through several menus. Even for new users, the choices may be easy, so that each option can be firmly rejected (or accepted) in turn, and not held in short-term memory.

Besides, if the seven item limit was true for computers, it would be true elsewhere - have you ever heard a restaurateur explaining the menu was kept short to avoid overloading customers' short-term memories? Restaurant menus are organised into courses, and meat dishes may be separated from fish ones and so on. We soon learn the conventions, and don't look for a chocolate pudding higher than the meat course.

It's equally important to organise big menus on computers. Palettes in a drawing or painting program can be easy to navigate - even when they're very large, such as on some CAD systems. These menus are set up in various ways, such as separating tool-selection items from pattern-selection ones, and similar patterns are usually grouped together.

There are even misguidelines on constructing menus. One is based on the assumption people search lists using the same 'algorithm' as simple programs. A system designer, embarrassed that he'd failed to observe the not-more-than-seven-options rule, proudly told me how he'd dealt with nearly 30 items in his pop-up menu. He'd put the most frequently used at the top. I gently pointed out that, with menus of that length, users try to learn the positions of the options and wouldn't always search from the top. The best way to help them is to structure the menu, putting together the six options on graphics editing, the five dealing with text labels and so on separating groups with lines.

Good structuring of menus not only speeds up searching and learning the positions of frequent commands, but helps you predict roughly where to find commands you've never used.

There's a misguideline which says that you should cater for all possible menu choices (itself a laudable objective) - *even if it creates overlapping choices between menus.* The flawed reasoning is: overlapping options doesn't matter, as both routes lead to the same place in the end. But new users reasonably assume they're being offered a real choice, and worry about getting it wrong.

Lists of good guidelines can be useful, especially when embedded in a discussion of issues that discourages blind and inappropriate application. See, for example, the chapter on Menu Selection Systems in Ben Shneiderman's book (details given in the reference box) for such a list and discussion.

I've given you a glimpse of what can go wrong with user interface design even when developers think they're observing human factors guidelines. If I'm to have any developers left as friends I'd better move on to users!

# TopSpeed to CeBit'91 with JPI

COMPETITION

Occupying a site the size of a small town and spread over 23 halls each focussing on some particular aspect of computing and IT, CeBit this year runs for 7 days from March 13th to the 20th. JPI, (Jensen & Partners International), in association with .EXE magazine, is offering you the chance to win a trip to this spectacular event.

CeBit, always the scene for new product launches, will this year set the scene for the launch of TopSpeed version 3.0, a significant milestone in PC applications development. JPI, producers of TopSpeed C and TopSpeed Modula-2, will be showing version 3.0 of the TopSpeed integrated multi-language development environment. As well as being able to mix and match C & Modula-2 coded, JPI will add TopSpeed Pascal, (an ISO compliant Pascal with Objects and JPI extensions), and TopSpeed C++, (AT&T2.1).

The two first prize winners will each receive a ticket for their flight to Hanover. A limousine will take you direct to the fair for a full days browsing all that's new in IT. After a preview of TopSpeed version 3.0, you will join Niels Jensen, Chairman of JPI for dinner. On return to the UK you will receive a complete TopSpeed Professional Developers Kit consisting of the TopSpeed integrated multi-language programming environment and all four TopSpeed languages. Winners may choose either TopSpeed for DOS or OS/2.

Ten runners up will each receive TopSpeed environments and their choice of either TopSpeed C++ or TopSpeed Pascal for either DOS or OS/2. A further 15 runners up will receive a .EXE T-Shirt of their choice and a .EXE mug.

Simply answer the five following questions and send your answers by post to the address shown alongside:

**Q1:** *Which is the only ANSI compliant C compiler for DOS based applications?*

**Q2:** *Which Modula-2 shares common pragma directives with three other TopSpeed languages?*

**Q3:** *What two languages are being added to the TopSpeed family of integrated compilers?*

**Q4:** *Which compilers are brewed in England by Danes?*

**Q5:** *Which programming and systems journal celebrates its 5th year in (February) '91?*

Name: .................................... Job Title: ....................................

Organisation/company: ....................................

Primary language used: ....................................

Compiler: ....................................

Operating System: ....................................

Address: ....................................

Phone: .................................... Fax: ....................................

*Winners will be drawn from correct entries received by March 10th. Flight winners will be notified by fax the same day. All other prize winners will be notified by post. The judges decision is final:- Post or fax your answers to TopSpeed to CeBit'91. 3 The Mansards, Tavistock Street, Bedford, MK40 2RX. Fax +44 (0)234 217094*

## Users

It's their own fault, really. Users are insufficiently demanding of good user interfaces. This is partly because people are adaptable and can cope with poor designs, even if at great cost in errors, stress, speed of work and training.

Amazingly often I find these costs go unnoticed, or it's assumed they're inevitably part of having a computer system. Rarely are there arrangements for spotting user problems. End-users with ideas for improvements seldom find ways of passing on the information through supervisors to decision-taking management.

A new client of mine did have regular meetings between senior end-users, management and software staff. Over the years many extra facilities and improvements had been provided in this way, mostly successfully. But even here there were problems. In one case, the user interface for a new facility clashed with the original design - causing extra user errors. Although it's vital to involve end-users in the design process, they're no experts on user-interface design.

More typically, end-users accept what they're given, blaming themselves for any difficulties. This is very common, and it's worth considering an example in detail.

A user was asked to demonstrate a sales enquiry system. Suddenly he swore, and gave a embarrassed apology that he'd made a mistake. I asked him to explain. 'It's just a silly slip I often make,' he said. 'All the more important to understand why', I replied.

He wanted to abandon a line of enquiry and return to the base screen. His error was to enter *A* (for *Abandon* this line of enquiry) instead of *R* (for *Return* to enquiry system). From all other lines of enquiry *Abandon* would have been correct, but he was using a screen not originally designed (but now often used) for such enquiries and here *A* meant *Abort*, causing him to lose even the customer's name and the nature of the enquiry.

The next six users I spoke to all admitted making the same mistake and blamed themselves for doing something they knew was wrong. They were told during training not to enter *A* at this point; asking customers to repeat their name and enquiry isn't good for customer relations. The operators didn't suspect the error was connected with their high level of skill.

As users become skilled, lower level actions become automatic subroutines executed with a single initiating thought or command. A skilled driver has only to think 'change gear'. A L-plated novice needs to think 'find clutch with left foot, press it down, move hand to gear-stick...' Automatic, skilled actions are smoother, quicker, and leave you free to think about higher levels of your task.

***With all this help and information available, there is no excuse for user abuse***

Skilled users of the enquiry system think 'Back to base screen' and an automatic subroutine runs off the habitual low-level actions, entering *A*. The fact that the same high-level action occasionally requires a different low-level one, entering *R*, is an inconsistency that's a trap for the skilled. Either the advantages of skill are lost, as users concentrate on the details of the task, or they revert to habit at the risk of errors.

## Abandon or Abort?

Investigating this led me to an identical result from a quite different cause. When the mainframe was busy and its response became slow, staff often typed ahead. On occasions, they would correctly enter *A* to abandon back to the base screen, but the computer wouldn't respond immediately. Assuming they had forgotten to enter *A*, they'd do it again. The computer would then use the first *A* to return to the base screen and the second *A* to abort - with the same consequences as before.

The system had many small traps like this, and a few more major problems. Senior staff had noticed a steady increase in the time it took to train new staff, but had no idea why.

What can we learn from this anecdote? First, users tend to blame themselves for making errors - and other difficulties - when much of the blame lies with the user-interface design. They're more likely to cover up errors than report them or discuss them with other users. Each user thinks that he's the only one having the problem, and nothing gets changed. So managers often don't know their staff are using a system that could be far better - with higher rates of work, fewer errors and users getting greater satisfaction from their work.

Second, it shows how system use can change over time, especially when new facilities are added. When new uses are made of existing facilities, the original design may be violated and new inconsistencies can interfere with the skills in unexpected and unnoticed ways.

Third, problems with slow responses go beyond time wasted waiting. Experienced and time-pressured users will try typing ahead and, with no indication of the computer's response, they may misinterpret what has happened and make inappropriate entries.

Slow responses are something users *do* notice, discuss and complain about. The enquiry system staff had done just that. So the computer department logged response times. The printouts showed responses rarely exceeded a few seconds, even during busy periods. The perception of slow responses didn't match the objective measurements and I was asked to investigate. One reason was easily found. To navigate across several screens, each step or scroll involved a transaction with the computer. Subjectively, the response took *several times* a few seconds - very noticeable when you're keeping a customer waiting.

So the users blamed the computer for slow responses, but themselves for errors. If they hadn't complained about the response times, I'd never have been called in, and the other problems would still be concealed. If users do become more demanding, where can they and the developers turn for help?

## The Experts

Certainly there is help available, but having taken both developers and users to task, it's only fair to put user-interface experts under the critical spotlight.

First, there are the would-be experts and not so experts. For example, people knowing little about the relevant subjects throw around the term *user friendly* to satisfy a client or beef up an advertisement, and now it's almost a term of abuse.

Potential experts - psychologists and ergonomics experts - were slow to apply their knowledge and skills to software. Even now, many experts are research-oriented rather than application-minded. Despite such research efforts, methods for designing user interfaces have been slow to appear, remain few in number and aren't widely

# "The MKS Toolkit is an amazingly faithful replication of a System V UNIX™ environment."

– UNIX Review

## Essential UNIX Tools for any PC user


*"The entire MKS documentation package proved excellent in every respect"*

Daniel McAuliffe, IEEE Computer, Jan. 1990

We can tell you that MKS Toolkit offers both experts and novices the purest form of UNIX utilities in a DOS or OS/2 environment. Fortunately, we don't have to. Software reviewers, universities and major corporations all over the world are discovering MKS Software and how it can help their programmers and software developers make the most of their creative talents.

With our proprietary code, MKS Toolkit offers you more than 150 UNIX System V. Release 4 compatible tools for DOS or OS/2. With MKS Toolkit, your computer or clone becomes a comfortable environment for shells, string matching, editing, file manipulation, and more.

In this one package you will receive a complete:

- MKS Vi full – screen editor
- MKS Korn Shell – user programmable interface
- MKS AWK – programming and prototyping language
- profiler – to analyse the execution time of code
- egrep/fgrep – fast pattern matching
- archivers, sort/merge, encryption, data compression & much more

Over 150 commands that help you get the most productivity out of your PC.

### Easy Installation

With MKS's menu-driven installation you can be up and running in less than 10 minutes, no matter what your level of computer expertise.

*"MKS software is absolutely the best in its class. Don't mix environments without it."*

Grover Righter, Novell

### Interconnectivity & Lan Licences

MKS Toolkit interacts well on most standard PC and PS/2 networks. Combined with Novell Netware™, the most popular LAN for PC's, MKS Toolkit creates a UNIX time sharing system in DOS or OS/2 organizations. UNIX workgroups can now hook up all of their PCs using PC-NFS™ and MKS Toolkit, enabling you to use a PC as a UNIX workstation and off-load your mini or main machine.

### Spanning Novice to Expert

This versatile software reflects it's users' needs. UNIX experts will have all the familiar power they demand while novices will find our documentation filled with excellent tutorial material.

Organizations such as AT&T, Hewlett-Packard, ITT, and NCR – all heavily committed to the UNIX system – use MKS Toolkit to create a standard operating environment. Universities from King's College to the University of Berlin use MKS Toolkit to enrich personal research computing environments and double the potential of their PC teaching labs.

*"With this package, you can become familiar with the UNIX environment on your microcomputer, with DOS only a keystroke away."*

Byte Magazine, May 1989

### Cost-Effective Learning Tool

If your organization is committed to moving into the UNIX environment, then MKS Toolkit is the perfect learning path. DOS or OS/2 users retain the familiar world of their PC keyboard and programs yet can move to a desktop UNIX environment at a keystroke. Exposure to new commands and functionality now becomes an integral part of the novice's working day.

### MKS Programming Platform

MKS Toolkit can be purchased as an individual product or you can buy MKS Programming Platform which contains MKS Toolkit and three other MKS products. MKS Platform helps smooth out the details of programming and software development by adding time-saving utilities such as: MKS RCS (Revision Control system), MKS Make (an automated program builder), MKS LEX & YACC (compiler learning and construction tools).

In all, you simply cannot find a more complete set of commands and utilities to get from DOS or OS/2 to UNIX. Once you have used MKS Toolkit, programming will never be the same again.

**US List Price**

| | |
|---|---|
| MKS Toolkit (DOS & OS/2) | $399.00 |
| MKS Toolkit (DOS/2 only) | $349.00 |
| MKS Toolkit (DOS only) | $249.00 |

**Please contact MKS dealers for UK domestic prices.**

The Software Construction Co. Ltd. (0763) 244114; fax (0763) 244025
Grey Matter (0364) 53499
Systems Science (071) 833-1022; fax (071) 837-6411

MKS

35 King Street North, Waterloo, Ontario, Canada N2J 2W9

Put The Power Of UNIX In Your PC.



CIRCLE NO. 485

known. Design tools for user interfaces were also slow to appear and are mainly confined to providing pre-programmed elements.

Sometimes the misguidelines that mislead designers have originated from research into human-computer interaction or psychology. One reason was over-interpretation of results of early experiments. Often later experiments produced apparently conflicting results when the context or measures were changed.

Some people assume that research on HCI is all about new methods of I/O. This is far from the case. There's still much to find out about basic topics, such as choosing command names, itself an area of apparently conflicting results.

For example, should command names be natural phrases, computer-oriented ones, or new words made up specially for the purpose? This whole area is much more complicated than even the researchers first thought. A review by Barnard and Grudin in the *Handbook of Human-Computer Interaction* shows the complexity behind what appear to be simple choices, but they also manage to dole out some sound advice based on our rather limited state of knowledge.

***Users blamed the computer for slow responses, but themselves for errors***

This can be repeated for many other topics; even the specialists don't have all the answers, but in most cases they do have *some* answers and can often give very relevant advice about how to proceed in difficult areas.

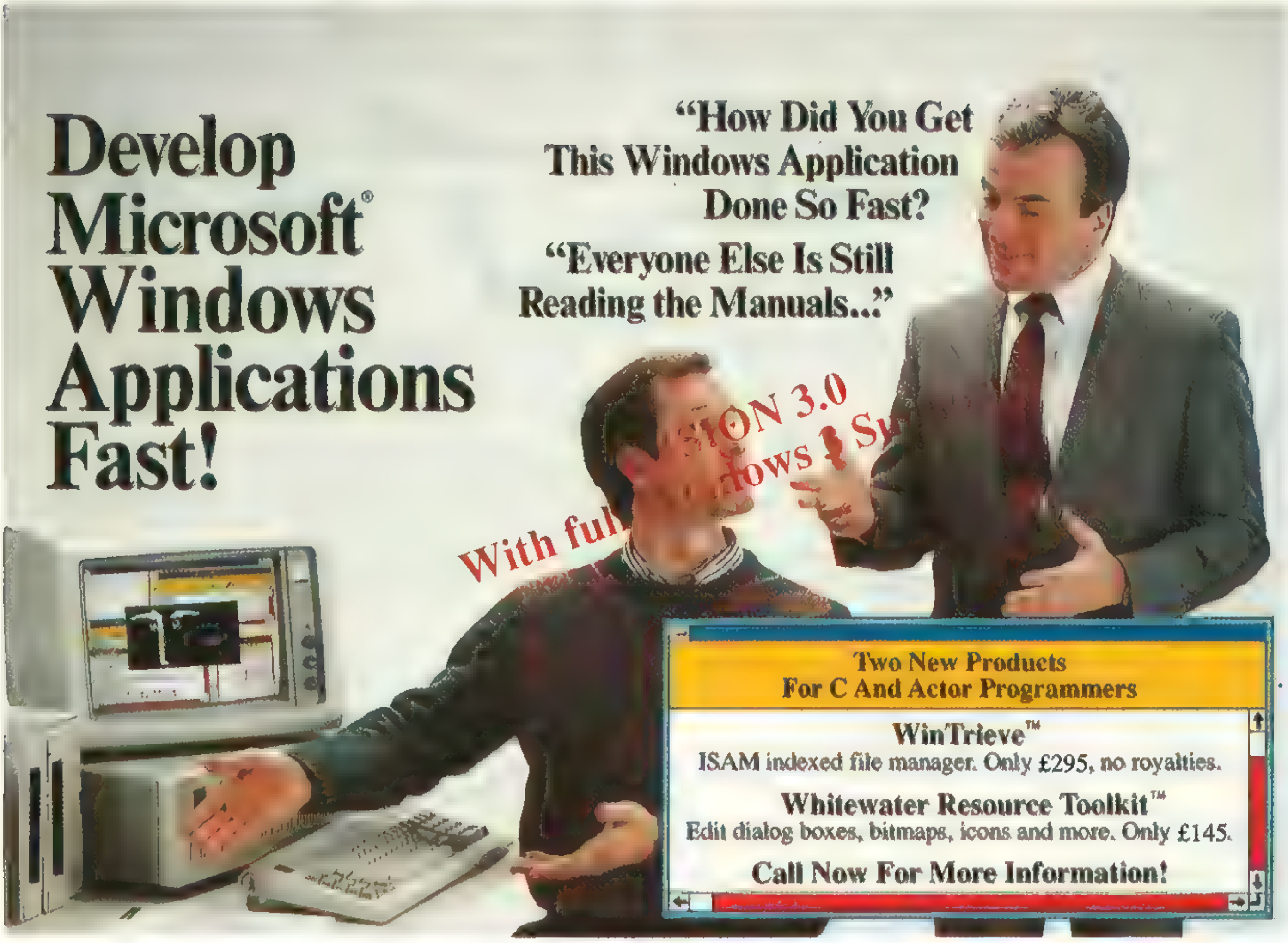
As important is the general approach taken by people whose first responsibilities are to understand the users and their tasks, and to act as their advocates in the design process. Some methods (eg task analysis) and tools (eg for prototyping) are already available.

## What can be done?

If progress is to be made, users will have to demand more usable systems. Developers, even the readers of .EXE, can't be expected to spend the time and effort needed if there's no demand. Users are often unaware of the problems, of their cost, of what can be done about them and of the availability of specialists.

Even the government seems worried. Instead of expecting the market to sort things out, it's putting DTI money into an HCI-awareness programme called *Usability Now*. Launched in November 1990, it includes guides, case studies, a newsletter, help with access to consultants and more. It aims to make users aware of usability issues, and help developers cope with the resulting demand for more usable systems.

User organisations will be encouraged to make sure developers have relevant expertise (eg for graphical user interfaces). As developers you can seek specialist help where necessary - but you too must check experts' competence. The Design Council publishes a listing of approved human factors consultants (reference box), though it isn't comprehensive.




## The Windows Advantage

You've selected Microsoft Windows as your graphical user interface. Windows applications are easier for end-users to learn and use. It is **the** standard for PCs.

Now you want Windows applications completed as fast as possible. But it takes time for your programmers to get up to speed on the intricacies of Windows development.

## Why The Delay For Windows Applications?

Programming for the Windows environment is very different from character-based programming. Just learning the 450 Windows function calls can take months. Getting used to its event-driven system and memory management takes additional time.

Programmers who are new to Windows development are looking for support to help them produce high-quality Windows programs quickly and efficiently.

## 3 Ways To Speed Up Windows Application Development

Now there are specific ways to speed up applications development under Windows.

- **Code reduction.** There is a fast, effective way to develop Windows applications. A way that cuts the amount of code needed. A way that by-passes much of the complexity of Windows programming. A way that doesn't compromise performance.
- **Short learning curve.** There is an easy, convenient way to get your entire programming staff up to speed with Windows within weeks.
- **From prototype to complete application.** There is a way to quickly develop prototypes of Windows applications in days, not months. Then, turn those into completed applications quickly.

## 1 Product, 3 Solutions

There is **one** product which brings **all three** solutions to your Windows programmers. It's called **Actor**®

**Actor** is a proven product. It's now being used in thousands of software development groups world-wide.

## Actor Does Windows

**Actor** gives your programmers a fast, effective way to develop Windows applications. **Actor's** complete development environment allows you to produce everything from prototypes through stand-alone applications. Programs developed in **Actor** are fast and support all Windows features.

**Actor** offers interactive source code debugging and testing—greatly reducing development time. Your programmers benefit from high code reusability because **Actor** is an object-oriented programming language. This allows them to build on existing code, eliminating duplication of effort. They can even dynamically link to existing C code.

**Actor** also provides one more essential: fast, responsive support from The Whitewater Group® recognized worldwide as leaders in Windows programming.

If you are programming applications for Windows, you should be using **Actor.** Users report that **Actor** cuts Windows programming time by at least 50%.

## Act Now!

See for yourself how **Actor** can help your Windows programmers! The first step is to order our free booklet, **For Faster, Better, Smaller, Smarter Windows Programming.** Examine detailed information about using **Actor,** complete with actual source code examples.

Call us at 0628-665722. We'll send you the booklet and complete details about **Actor.**

Take 60 seconds now to save months of development time in the future.

***NEOW*** **- The Windows Specialists**

Unit 12, Progress Business Centre,
Whittle Parkway, Slough. SL1 6DQ.
Telephone: 06286 68334. Fax: 06286 65722



CIRCLE NO. 489

Don't feel threatened by more knowledgeable users. Many development problems occur because of a lack of understanding between users and developers. It's important to involve users, including end-users, in many aspects of development. Investigate user issues with them at the requirements stage, not later. Prototype the user interface and let end-users try it out.

It's helpful if your users understand the stages of development, their own roles and those of the developers. Refer them to *Developing Systems Together* (reference box), a booklet that explains all these in non-technical language with the aid of cartoons; it even has a short chapter on the user interface.

If you want to learn more yourself, there are several channels. Consider possibilities for training in HCI; for example, there's a new Open University course. There are some good and readable HCI textbooks - but be careful not to get bogged down in the large 'researchy' tomes written for specialists by fellow researchers.

For specialists there are other information sources, such as the HCI literature service run by the Scottish HCI Centre. The HCI Specialist Group of the British Computer Society offers meetings and a newsletter.

With all this help and information available, there is no excuse for user abuse!

EXE

## Bibliography and Reference

*Designing the User Interface*
Ben Shneiderman, Addison-Wesley, 1987, ISBN: 0-201-16505-8, £17.95.

*Handbook of Human-Computer Interaction*
M Helander (ed), Elsevier Science Publishers, 1990, ISBN: 0-444-88673-7, £56.40

*Developing Systems Together*
Available direct from the National Computing Centre, Oxford Road, Manchester M1 7ED, (061-228 6333), £30 for 10 copies

*Directory of Designers* (see section on human factors)
The Design Council, 28 Haymarket, London SW1Y 4SU, ISBN: 0-85072 275 6, £25.00

Open University course on *Human-Computer Interaction*
PMT 607: Information on 0908 652917

The *Computer-Human Interaction Collection*
Scottish HCI Centre: enquiries on 041-552 1576

*British Computer Society HCI Specialist Group:*
Enquiries to the BCS on 071-637 0471 - ask for specialist group liaison.

*Dr John E Harris runs HCI consultancy, CAMBIT The User Interface (0223 811487). He is a Chartered Psychologist with a background of research in cognitive psychology at the Medical Research Council, two years with Scicon, followed by Alvey Project work and user documentation.*

# Meeting the deadlines

*The real time executive is the heart of a fast real time system, and the subject of volumes of academic research. But Jim Cooling is not impressed.*

Electronics has played a major part in the control, monitoring and instrumentation of 'real world' systems for many decades now (the Centurion tank gun stabiliser, for instance, first saw the light of day about 45 years ago). Three particular features were originally identified as essential requirements of these systems: reliability, safety and deterministic behaviour.

There was, in reality, a fourth requirement: speed of response. Yet for the analogue (and, later, the digital) designer, this was a non-issue. Electronic controls were so fast (compared with the system being controlled) that speed was not a problem. Another feature marked out these real world systems; the electronics formed an integral part of the **total** system. That is, their design and function didn't make sense taken in isolation from the application.

When the microprocessor arrived, circa 1970, it seemed (with its power, flexibility and programmability) like the ideal device for future systems... until we encountered the Von Neumann bottle-neck. Suddenly response times **were** an issue - and an important one at that. It was clear that real time systems had definite, often rigid, deadlines. Failure to meet these produced unacceptable - sometimes catastrophic - results. Computing speeds, it turned out, were a crucial factor in 'real world' applications.

| | SLOW | FAST |
|---|---|---|
| SOFT | ON-LINE INTERACTIVE SYSTEMS | CAE GRAPHICS SYSTEM |
| HARD | SLOW CHEMICAL PROCESS PLANT | FLIGHT CONTROL COMPUTER |

*Figure 1 - Categorising Real-Time Systems*

***All the work was done on a VAX 11/784. Very useful for people working with microcontrollers***

By the late 1980s, conventional electronics had, in the main, been replaced by computer techniques. To distinguish these reliable, deterministic and timely systems from, say, on-line computing, a new name was coined: 'hard real time embedded systems'. Now, does this also imply that hard real time systems are fast? Many would say no. Their argument is that any system having *definite* deadlines falls into this category - *actual* response times are not important. Unfortunately, if this definition is used, the practical difficulties of meeting fast deadlines is often ignored (certainly true of much current research literature). I believe that real time systems should now be categorises into four groups, as shown in Figure 1.

The boundary between 'fast' and 'slow' is obviously subjective (here is a personal view: where response times are measured in seconds, systems are slow). In contrast, the distinction between soft and hard responses is very clear-cut. Soft systems - no matter how fast they are - can go late on their deadlines without causing problems. This is most definitely not true of hard applications.

## The fast response

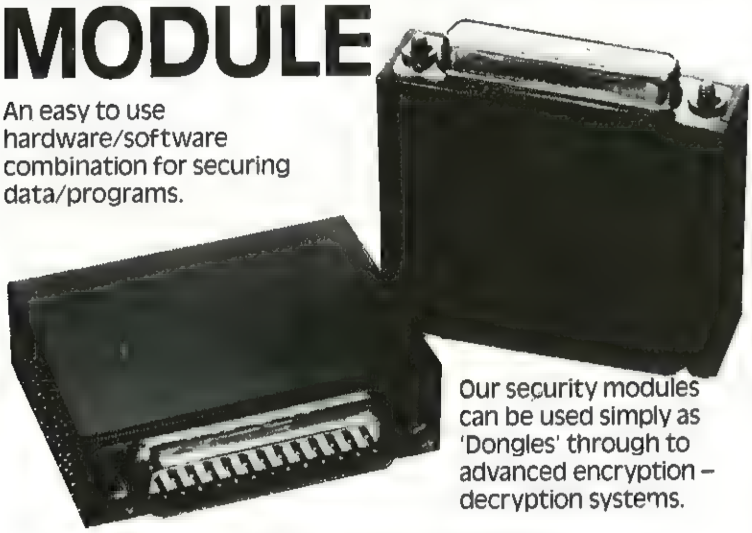
How can you guarantee fast responses from computers? Probably the most widely used method is to make time-critical tasks interrupt-driven. Where a system contains only one such task this is very easy to implement - and be confident of meeting the deadline. However, few embedded systems are this simple; most consist of several time-critical processes. For example, the flying laboratory system fitted to a Canadian research aircraft1 contains (among others) the software processes shown in Figure 2.

Moreover, it isn't unusual to find that while one task is being processed, others demand service. To control processor behaviour in these circumstances, the programmer normally assigns priorities to tasks (in the Figure 2 example, the ordering is by priority, with 1 being highest). In general, a running task may be replaced ('pre-empted') by a higher priority one which requires service (though tasks may be blocked out by disabling interrupts). The problem is: when using this technique, how can you be sure that all tasks will meet their deadlines? In small systems, tasks are visible enough for the programmer to estimate accurately system responsiveness. But what about large systems? And what about designs based on true multi-tasking software, running under the control of a real time executive?

Modern executives designed for real time embedded systems (such as Ready Systems VRTX32 and Software Components Group pSOS) use priority task scheduling schemes. The difficulty here is to decide on the 'best' way of setting schedules. A very common approach is to assign priorities at compile time, leaving them fixed thereafter (a static schedule). Priorities may also be

# SOFTWARE SECURITY MODULE

An easy to use hardware/software combination for securing data/programs.



Our security modules can be used simply as 'Dongles' through to advanced encryption – decryption systems.

- ★ For IBM PC family (and compatibles)
- ★ Variants for other systems
- ★ Advanced construction – rugged and compact
- ★ Proven throughout the world
- ★ Transparent operation – user friendly
- ★ Driver software supplied for variety of applications

CTL

**Control Telemetry of London**
**11 Canfield Place, London NW6 3BT**
**Tel: 071-328 1155 Fax: 071-328 9149**

CIRCLE NO. 491

# PROTECT YOUR PROGRAM FROM PREDATORS

- Established for over eight years in software security products
- Internationally established product
- Competitively priced keys (from £18.00)
- Supports IBM PC/XT/AT and PS/2
- Supports Apple Macintosh range
- Advanced manufacturing techniques
- Transparent and user friendly
- No drivers necessary (with exception of programmable key)
- Large range of security keys.

*For further information and 60 day evaluation kit, contact:*

**COMSEC (U.K.) LTD.**
**Cranfield Airport,**
**Cranfield,**
**Bedfordshire,**
**MK43 0AL**
**Tel: (0234) 751203**
**FAX: (0234) 751082**

COMSEC
(U.K.) LIMITED
COMPUTER & COMMUNICATIONS SECURITY SYSTEMS

CIRCLE NO. 492

# System Science

## *Microsoft* Languages

### C Professional Development

The latest C 6.0 has improved optimisation, in-line assembler and draft ANSI support amongst other enhancements. The new Programmer's Workbench integrates the operation of the compiler, make, linker code browser and the new Microsoft C Advisor. CodeView 3.0 can reside in extended memory for DOS. Dos and OS/2 support is included for just........... **£245.00**

### Basic Professional Development

New version 7.0 handles large applications with run-time overlay support, far strings and EMS4.0 support. New included toolboxes for menus and user interfaces, presentation graphics and maths. New ISAM for transaction processing and a data dictionary. The Microsoft Basic advisor is in the Extended Quick Basic environment. DOS and OS/2 ....**£275.00**

### Windows Software Dev. Kit 3.0

Windows SDK provides a set of tools and documentation for developing applications for the new Windows 3.0, including a resource compiler, dialog editor and libraries for DDE (dynamic data exchange). Use it with Microsoft C 6.0 or 5.1 to put the most into your Windows app. ...............................**£325.00**

**Windows 3.0** ..............................................................**£89.00**

### Fortran-77 Compiler ver 5.0

Fortran 77 with VAX and proposed 8X extensions, the compiler supports large arrays, large data and large code and graphics primitives. Editor, linker and CodeView, the source code debugger,are included. For DOS and OS/2 .....**£245.00**

### OS/2 PM Toolkit

For OS/2 ver 1.2, the PM toolkit contains extensive documentation on OS/2 and PM programmming, with software tools, such as the Icon Editor, Dialog Box Editor and resource compiler, with the on-line Quick Help system. Use the toolkit with C, Fortran or Basic ..............................................**£325.00**

### Further Microsoft Products

**Assembler 5.1 (DOS and OS/2)**.......................**£95.00**
**Quick C ver 2.5** ...................... .......................**£65.00**
**Quick C with Assembler ver 2.51**.......................**£98.00**
**Quick Basic ver 4.5** ..........................................**£65.00**
**QuickPascal ver 1.0** ..........................................**£65.00**
**MS Cobol ver 3.0 (Dos and OS/2)** ................**£475.00**
**Microsoft Programmers Library (CD-ROM)**...**£275.00**

**3-5 Cynthia Street,**
**London N1-9JF**
**Fax: 071 837 6411**

VISA

**Prices are exclusive of VAT, but do include UK delivery.**
**Prices are subject to change and alteration.**

# 071- 833 1022

CIRCLE NO. 493

changed using program instructions (dynamic scheduling); clearly this technique must be used with great care.

This scheduling policy has two major drawbacks when used in hard real time systems. First, the programmer is totally responsible for ensuring that deadlines are met. Second (and very much a consequence of the first), processor utilisation is often quite low. A most unsatisfactory state of affairs - but how can the situation be improved?

## The Future

In the future, scheduling policies must be developed that eliminate the need for programmer intervention, improve processor utilisation and ensure that deadlines are met.

We are going to see both short-term and long-term solutions to these requirements. The long-term ones will be generated by the computer science community - although current research papers suggest that practical schemes will be a *long* time in coming. The realities of hard real time embedded systems - size, weight, environment and response times - don't seem to be well understood by academics.

For instance, a paper entitled *Implementation of a time-driven scheduler for real time operating systems* looks inspiring - until it turns out that all the work is done on a VAX 11/784. Very useful to people working with microcontrollers. And *Imprecise results: utilising partial computations in real time systems* turns out to have been developed on a set of networked Sun workstations. I can't quite see the European Fighter Aircraft hauling that lot about the sky, even if processors are becoming more powerful. A paper entitled *Implementation and evaluation of a time-driven scheduling processor* contains the following illuminating statement: 'For the loads used in our experiments, the average task processing time was 0.5 seconds per request. This was usually sufficient to avoid host processor idle periods, awaiting the outcome of occasional long scheduling decisions'. With response times like that on offer, you don't need a computer - you can do better all by yourself. Some good work is going on, but it's going to be quite some time before it becomes usable.

Short-term solutions will (and in some cases, already have) come from the suppliers of real time executives. Commercial pressures are forcing the pace, even if it's only a matter of keeping up with competitors. Emerging systems may not be perfect, but they do hold potential for the future. Two basic scheduling techniques are offered: 'rate monotonic' and 'deadline driven'. A combination of these, 'mixed' scheduling, is also likely to become another practical scheme.

***Some important modifications had to be made to overcome drawbacks in Ada***

## Rate monotonic

Rate monotonic scheduling is simple and straightforward - but applies only to periodic ('synchronous') tasks. Here, tasks are assigned priorities which depend on their repetition rates (frequency). The highest frequency task gets highest priority, lower frequency ones having successively lower priorities. The one with the lowest repetition rate ends up with the lowest priority in the system. Tasks are then run under a pre-emptive scheduling policy, as shown in Figure 3.

Note that no attention is paid to task execution times when setting priorities. Note also that, as defined here, a task's deadline is the time interval between successive activations.

This technique was analysed in detail by Liu and Leyland[2] in 1973, who developed a measure called *processor utilisation* which is often quoted in relationship to rate monotonic scheduling. This is used to calculate whether tasks can be guaranteed to meet their deadlines when using rate monotonic scheduling. However, be warned; 'utilisation' is a misleading term. A set of tasks running on a single processor, using all the available processor time without missing deadlines, can have a calculated utilisation factor varying between 0.693 and 1.0.

Rate monotonic scheduling has been implemented by DDC International of Denmark[3] as an extension to its Ada run-time system. The Ada tasking model is a synchronous one, so this scheduling scheme fits in naturally with Ada programming. Some important modifications had to be made to the technique to make it effective in practice (NB: these were needed to overcome drawbacks in Ada, not in rate monotonic scheduling!). The problems and DDC's solutions are:

1) Tasks waiting for a rendezvous are held in a FIFO queue. A low priority task delays a higher priority one further down the queue. *Solution: use priority based queues.*

2) During task rendezvous, low priority tasks can block the execution of higher priority ones ('priority inversion'). *Solution: use priority inheritance. If a task is blocked by a lower priority one, the blocking task inherits the priority of the blocked task.*

3) The `Select` statement selects executable tasks in a random fashion. No attention is paid to priorities. *Solution: Choose the highest priority task within the* `Select` *operation.*

DDC says, 'measurements have shown that processor utilisation can exceed 90% in some systems before critical deadlines are missed, where only 30% processor utilisation was achieved without rate monotonic scheduling'.

Even so, there are a number of major weaknesses with the rate monotonic method. First, it applies only to periodic tasks. As can be seen from Figure 2, real systems involve both periodic and aperiodic (asynchronous) processes. Second, it does not take task importance into account. In reality, a task at any repetition rate may require an urgent response once it is activated. Third, the definition and use of deadlines is not good enough for many applications. Provided a task finishes before it is next activated, it is considered to have met its timing requirements. Thus all the responses shown in Figure 3 are, by this definition, satisfactory. Yet the task elapsed time for task B is considerably different in Figures 3b and 3c. In many situations (for example, a marginally stable closed-loop control system), this variation would be unacceptable.

| | PROCESS NAME | RATE |
|---|---|---|
| 1. | Urgent command processor | Asynch |
| 2. | Magnetic field counter | 8 Hz. |
| 3. | System process | 2 Hz. |
| 4. | Loran_C receiver | 0.1 Hz. |
| 5. | Pilot's command processor | Asynch |

*Figure 2 - Process Timing Example*

# dBase III Plus, FoxBASE & Clipper applications running on VAX & UNIX?


## Try our 30-day software licence
## In an hour or two you'll be converted

Can Recital really be that easy?

The answer is yes. And to prove it, we invite you to send for our full-featured 30-day software licence for VAX/VMS™ and UNIX systems. Recital is so compatible and easy to use, you'll have your dBase™ applications up and running in Recital in about an hour. Without any additional programming or modifications.

Then take the rest of the month to discover the richness and power of this 4GL system. Recital includes tools like a report writer, screen painter, transparent access to RMS files, an integral data dictionary, popup calendars, calculators, pick lists, note pads and much more. All on a VT-type terminal!

Your 30-day license will include the full Recital system, documentation and hotline support. How can you lose?

Order today by calling one of the numbers below. Or write to Recital Corporation.

Recital Corporation Limited
South Bank Technopark
90 London Road
London SE1 6LN, UK
Tel 071-401 2727
Fax 071-633 9617

Recital
CORPORATION

All trademarks are the properties of their respective companies

Recital Corporation Inc
85 Constitution Lane
Danvers
MA 01923, USA
Tel (508) 750 1066
Fax (508) 750 8097

CIRCLE NO. 495

## Deadline scheduling

The limitations of rate monotonic methods led researchers to look at dynamic techniques for scheduling tasks. The most promising turned out to be what is called *deadline scheduling.* Again, Liu and Leyland had a big hand in this. However, they stuck with their original definition of 'deadline' which really wasn't satisfactory. A much more useful measure - and highly applicable to real time systems - is the permissible delay between invoking a task and having it completed. We'll call this the task *due-time.* Further, our definition of a task deadline now becomes 'the time at which the due-time expires'. Within this system the important timing functions are:

(a) Task execution time.

(b) Required response time.

(c) Activation time.

(d) Elapsed time from activation.

(e) Time to deadline.

(f) Amount of task execution currently completely.

(g) Spare processing time [that is, (e) minus (f)].

The purpose of modern deadline schedulers is to ensure that system tasks are completed by their due-times. To set a task's due-time, we have to allow for three factors:

- How quickly a task needs service.
- How critical a task is.
- The actual task execution time.

Note that task repetition rates don't enter into the equation.

Each task has an initial low-level priority value, which may be set at compile time. When a task is invoked, it initially runs at this value. However, as it gets closer to its due-time, the priority is raised, as shown in Figure 4. Once the task is completed, its reverts to its initial value.

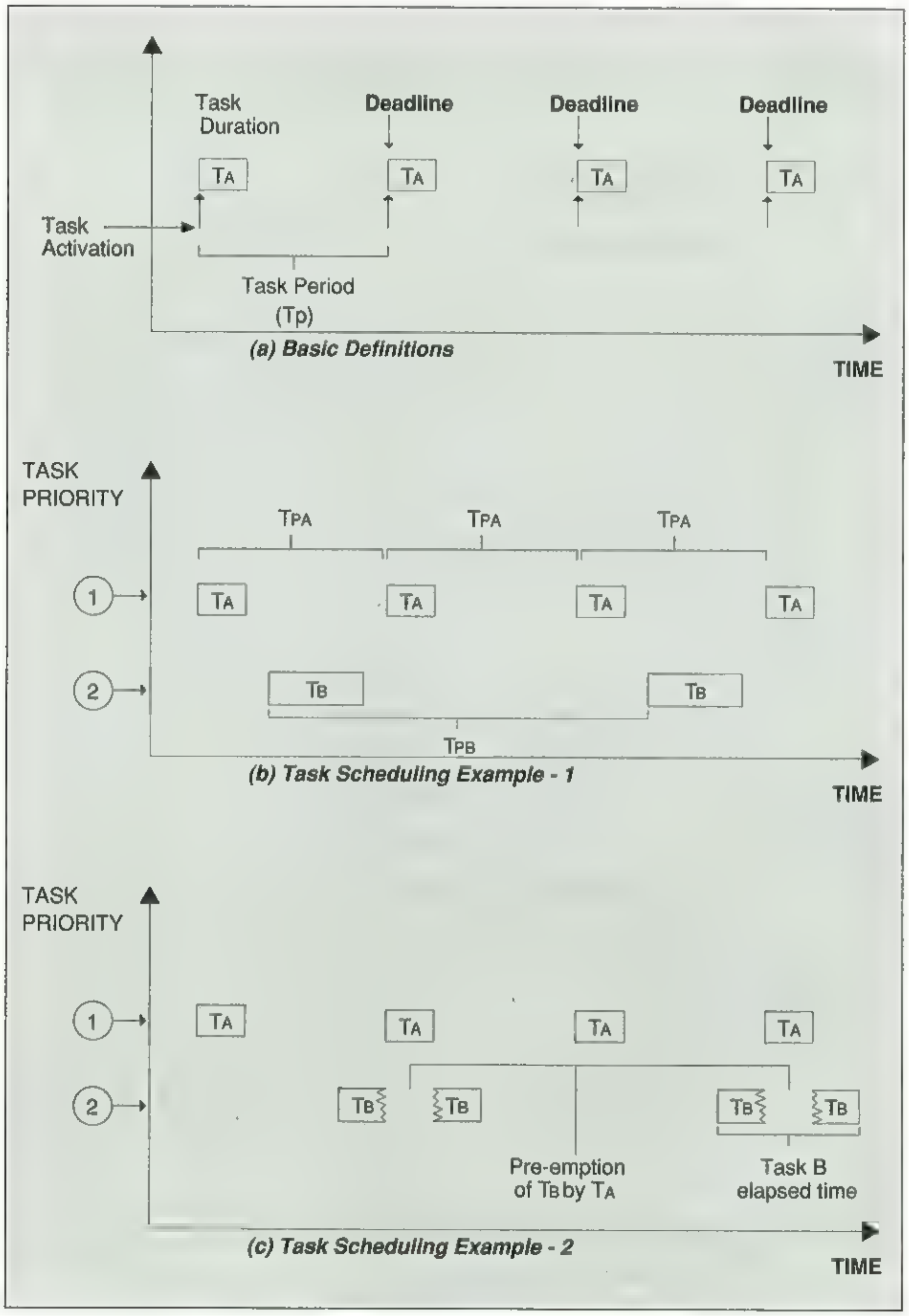



*Figure 3 - Rate Monotonic Scheduling in Action*

When using a simple implementation of deadline scheduling, the task closest to its deadline has the highest priority. The one furthest away has the lowest. The task scheduler always sets the highest priority ready-task running at reschedule time.

At first sight this seems to have cracked it - but there is no such thing as a perfect solution. Even the simple approach described here has one major problem: time. Deadline scheduling is a complex task. Implementing it in software will add substantially to the time overhead of the executive. In a soft real time system there should be enough processor time to run the scheduler in software. But what about fast systems?

In my view, the only way to meet such requirements is to off-load the scheduler into silicon - ie use co-processing. This is how the PORTOS ('Portable Real Time Operating System') deadline driven scheduling system[4] is built. With this system, timing factors are included in two ways. Items (a) and (b) noted above are defined at compile time. Items (c) to (g) are handled by an ASIC specifically designed for the PORTOS system.

At the present time, PORTOS is available for IBM PCs, single board computers (eg the Intel SBC 286/310) and embedded systems based on Intel microprocessors.

## Problems to be solved

The DDC and PORTOS executives represent significant steps in the development of real time executives. But they don't have all the answers to the needs of multi-tasking for real world applications. The next stage would seem to be the combination of deadline and rate monotonic techniques - 'mixed scheduling'. Even here there are

a number of thorny problems to be tackled, all of them inter-linked. The major ones are:

(a) Defining the 'best' task priority profiles.

(b) Linking criticality with profile.

(c) Handling overloading.

(d) Predicting performance.

Many priority profiles have been proposed for deadline scheduling; few have been implemented. There is really very little experience in this area. Further, complex profiles may prove to be great in theory and simulation - but hopelessly impractical for real time work.

A second issue is that of criticality. Suppose that serial data arriving in a UART must be collected within a set time period; otherwise the data will be lost or corrupted. Further suppose that a task is being executed which has a longer deadline than the UART task - only it must *not* be pre-empted for reasons of safety or security. How should this be handled?

This leads to the question of overloading, that is, tasks missing deadlines because of processor loading. Proposed solutions to this problem are based on the idea of executing critical tasks (which can then be guaranteed to meet their deadlines) and dropping or deferring others - *load shedding*. It is not currently clear how can we implement these effectively and safely, **and** within the time and performance constraints of real time systems?

Finally, a method is required for predicting the performance of application programs at the design stage. And, even more important, a technique is needed to measure and record the *actual* performance of the software within the target system.

## Multi-processors

Operating systems for hard real time multiprocessor systems are in their infancy. The overall subject is currently a 'hot' one in the research community. Unfortunately, much of the work seems far removed from the world of real systems; surveying the research publications of the last few years only induces great gloom. Not only is much of it verbose, pedantic and verging on the incomprehensible; it also frequently fails to address some really important questions. For instance, many researchers are heavily into 'load balancing' - the concept of shifting tasks from processor to processor to meet scheduling requirements. Load balancing algorithm development is top of the agenda. Yet questions such as:

- How do we develop the software for such schemes?
- How do we keep track of software execution?
- Do we have the time to evaluate loading and redistribute tasks?
- Do we even have the necessary information to do this?
- How do we ensure correct and timely inter-processor communication?
- How do we handle and recover from communication errors?

are rarely raised. We have a long way to go before the multi-processor problem is solved.

## Conclusion

It seems clear that, in the near future, scheduling methods for hard real time systems will generally remain unchanged. Current fixed/dynamic priority based techniques will continue to be widely used. However, rate monotonic and deadline scheduling systems will be gradually introduced, together with combinations of these. But

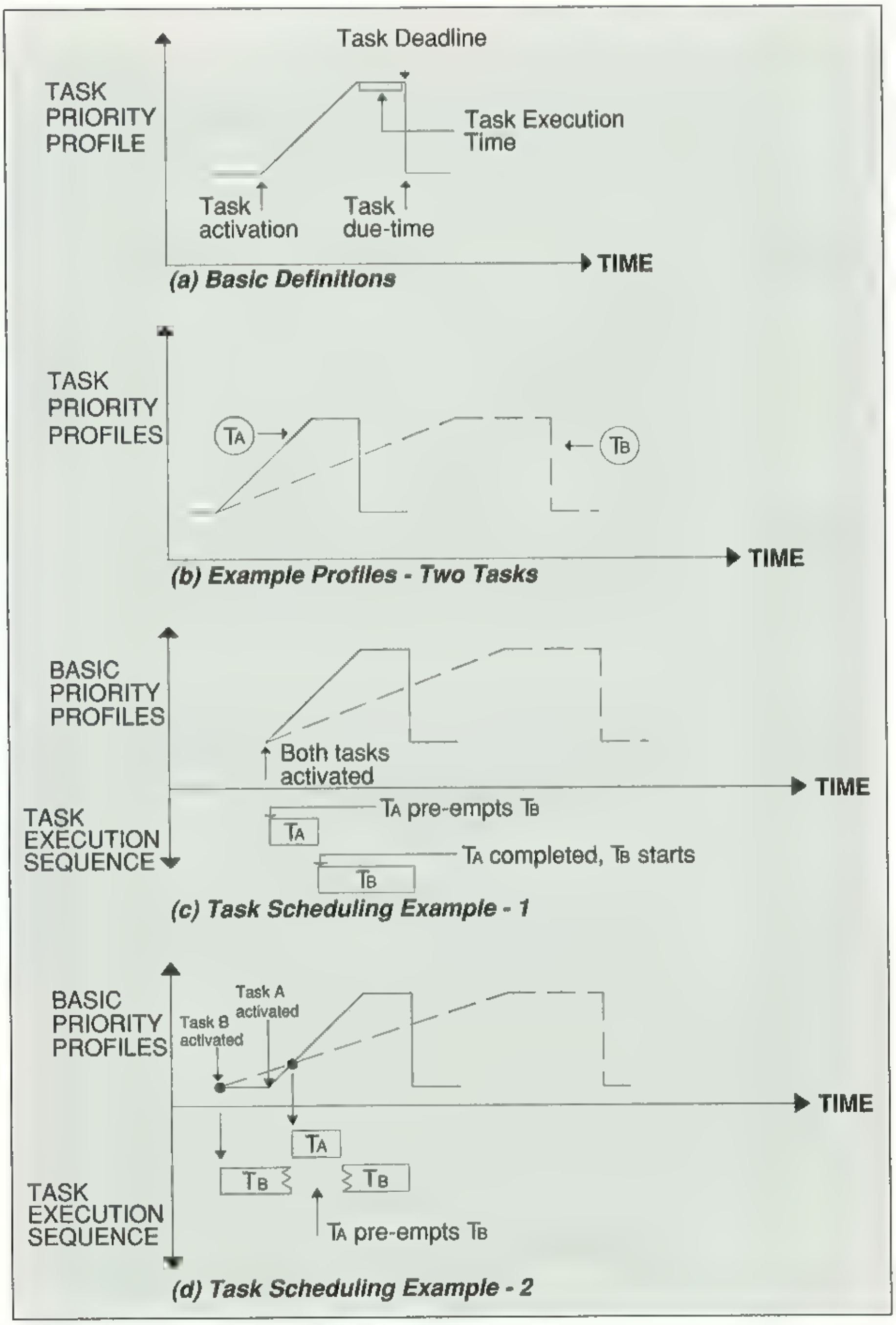



*Figure 4 - Deadline Scheduling in Action*

## REFERENCES

1. JE Jordan, *Experiences structuring software in a periodic real time environment*, Software Practice and Experience, Vol 20(7), pp 707-718.

2. CL Liu and JW Leyland (1973), *Scheduling Algorithms for Multi-programming in a Hard Real-Time Environment*, Journal of the ACM, 20(1):46:61, 1973, pp 174-189.

3. P Molin (1990), *Implementation of an Ada run-time system supporting hard deadline scheduling*, DDC International A/S, Gl. Lundtoftevej 1B, DK-2800 Lyngby, Denmark.

4. H Piche (1988), *PORTOS - Nutzerhandbuch der PORTOS-algorithmen - Antwortzeitgesteuerte Prozessorzuteilung* (= 'PORTOS - User's handbook of PORTOS Algorithms - Response time controlled by coprocessor'), Report GPP/442/88, GPP mbH, Oberhaching.


these need to be backed up by other software tools: comprehensive development environments, debuggers for real time concurrent programs and performance analysis tools.

***Research work seems far removed from the real world - recent publications induce great gloom***

More specifically, we need facilities to let us build multi-tasking software from the problem point of view. The approach should be based on graphics. You should be able to describe systems as sets of concurrent tasks together with their communication and interaction features. Timing, criticality, overloading and error-handling need to be spelt out. Integrated host/target development systems need to be produced. These should support design simulation, debugging and performance prediction using host features. For software development in the target system, a multi-tasking debugger is essential, augmented by performance monitoring and analysis tools.

This may seem like a giant wish list. But I believe that, over the next five years, improvements here will be more important than advances in executive design.

EXE


*Dr Jim Cooling is a senior lecturer in the Department of Electronic and Electrical Engineering, Loughborough University of Technology. He has been deeply involved in the design, development and production of real time systems for many years, and has published regularly on the subject.*



*Acknowledgement: The idea of defining new categories of real time systems was inspired by comments from Nick Whitehead and Niall Cooling of Ready Systems (UK) Ltd.*

# WE GUARANTEE Y
# WIDER RANGE OF SO

## Over 10,000 Software Products

● COMMUNICATIONS ● DATABASES ● dBASE ADD-ON'S ● DEBUGGERS ● DEMO'S & PROTOTYPING ● DESKTOP PUBLISHING ● DIAGNOSTICS ● DOCUMENTORS ● E-MAIL ● EDITORS ● EDUCA

LOTUS 1-2-3 ● 386/486 PRODUCTS ● ACCOUNTING ● C LANGUAGE ● CAD/CAM ● CASE & FLOW ● CD-ROM ● CLIP-ART ● CLIPPER

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| @Base | 145.00 | Close-Up/Customer | 139.00 | dSalvage Professional | 149.00 | Greenleaf Communications | 239.00 | LotusWorks |
| 386MAX Pro | 79.00 | Close-Up/Support | 169.00 | Dungeon Master | 24.99 | Greenleaf SuperFunctions | 179.00 | Mace Utilities '90 |
| ABC Flowcharter/Windows | 199.00 | CO/Session | 159.00 | Duplicator Toolkit | 69.00 | Gray F/X | 299.00 | Manifest |
| Above Disk Plus | 65.00 | CodeBase 4 | 195.00 | Dynacomm/Windows | 279.00 | HALO '88 | 329.00 | Map Assist |
| Advanced Revelation | 595.00 | Coherent/PC | 125.00 | EasyCASE Plus | 199.00 | Harvard Graphics | 279.00 | MathCAD |
| Allways 1-2-3/Symphony | 99.00 | Colorix VGA Paint | 125.00 | Easyflow | 149.00 | Harvard Project Manager | 495.00 | MathCAD Add-On's |
| Alpha Four | 349.00 | Command Plus | 79.00 | Edix | 129.00 | hDC FirstApps | 59.00 | Mathematica/386 w/387 |
| Ami Pro/Windows | 325.00 | Control Room | 79.00 | EMACS/DOS | 239.00 | hDC Windows Express | 59.00 | Mavis Beacon Typing |
| AmiDIAG | 75.00 | Copy II PC | 27.00 | EMACS/UNIX | 349.00 | Headroom | 75.00 | Media Master Plus |
| Animator | 269.00 | Copywrite w/Zerodisk | 53.00 | EnerGraphic. | 249.00 | Hi-Screen/XL | 99.00 | Memory Mate |
| Automate/Anywhere | 129.00 | Corel Draw/Windows | 279.00 | Essential Graphics | 239.00 | Hijaak | 89.00 | Menugen |
| Applause II | 279.00 | Corporate Ladder | 69.00 | EXP | 269.00 | Hooray for Henrietta | 19.95 | MenuWorks |
| Arkanoid II | 29.00 | Crosstalk Mark IV | 109.00 | Express Publisher | 129.00 | Hot Wire | 99.00 | MetaWindow Plus |
| Arts & Letters/Windows | 395.00 | Crosstalk/Windows | 97.00 | Extend-a-Name Plus | 73.00 | Hotshot Graphics | 159.00 | MicroTSP |
| AskSam | 185.00 | Cruise Control | 43.00 | Extra-K/Symphony | 69.00 | HyperAccess/5 | 169.00 | Milestones, Etc/Windows |
| Asyst Complete | 1995.00 | Dan Bricklin's Demo II | 159.00 | FaceLift/Windows | 79.00 | Hyperpad | 99.00 | Mirror III |
| ATI/Training | 59.00 | DASM | 169.00 | Family Tree Maker | 49.00 | IBM Current/Windows | 295.00 | MKS Toolkit |
| Auto-Might | 59.00 | Data Junction Advanced | 229.00 | Fantavision | 57.00 | IBM Displaywrite IV/V | 289.00 | Monarch |
| AutoImport | 129.00 | Dataease | 429.00 | Fastback Plus | 99.00 | IBM DOS 3.x | 89.00 | Move 'Em |
| Automenu | 45.00 | Dataease LAN Pack | 569.00 | Fastlock | 47.00 | IBM DOS 4.x | 99.00 | MS BASIC Compiler |
| Autoroute Plus | 215.00 | DataPerfect | 289.00 | Fastlynx | 85.00 | IBM Fixed Disk Organizer | 47.00 | MS C Compiler |
| AutoSketch v3 | 79.00 | dBASE III Plus | 327.00 | FastText Search/Clipper | 79.00 | IBM OS/2 Standard Edition | 239.00 | MS COBOL Compiler |
| Avery LabelPro/Lasers | 89.00 | dBASE IV/Standard | 369.00 | FastTrax | 45.00 | IBM Storyboard Plus | 229.00 | MS Excel/Windows |
| B-Tree Filer | 69.00 | dBFast/Windows | 329.00 | File Rescue Plus | 37.00 | IBM Writing Assistant | 129.00 | MS Flight Simulator |
| Baler | 310.00 | dBrief | 79.00 | Flowcharting 3 | 149.00 | Illustrator/Windows | 289.00 | MS F/S Scenery Disks |
| Batcom | 45.00 | dBXL Diamond w/R&R | 149.00 | Folio Views | 379.00 | Impress | 62.00 | MS FORTRAN Compiler |
| Battery Watch | 32.00 | Deluxe Paint/Enhanced | 79.00 | Fontasy 3 Deluxe | 79.00 | Individual/Training | 59.00 | MS Macro Assembler |
| BeckerTools/Windows | 69.00 | Derive | 125.00 | Fontrix | 129.00 | Info-XL | 149.00 | MS Mouse + Paint |
| Bitstream Fonts (each) | 135.00 | DesignCAD/2D | 169.00 | Fontspace | 69.00 | Info Select | 85.00 | MS Multiplan |
| Blast/PC | 159.00 | DesignCAD/3D | 229.00 | Formtool Gold | 63.00 | Instant Recall | 95.00 | MS Pascal Compiler |
| Blinker | 199.00 | Designer/Windows | 399.00 | Formworx/Windows | 249.00 | Instaplan 5000 | 449.00 | MS Powerpoint/Windows |
| Blue MAX | 89.00 | Desk-Link | 95.00 | Formworx w/Fill & File | 99.00 | Intelligent Backup | 95.00 | MS Project/Windows |
| Brainmaker | 175.00 | DESKpress | 229.00 | FoxGraph | 225.00 | Kedit | 109.00 | MS Quick C |
| Brief Editor/DOS or OS/2 | 179.00 | DESQview | 79.00 | FoxPro | 429.00 | Kermit for DOS | 39.00 | MS Quick Pascal |
| Brooklyn Bridge | 79.00 | DESQview/386 | 99.00 | FoxPro/LAN | 599.00 | KnowledgePRO/Windows | 529.00 | MS QuickBASIC |
| Btrieve for DOS | 389.00 | dGE Graphics | 229.00 | Fractools | 49.00 | Labels Unlimited | 42.00 | MS Windows SDK |
| Builder | 99.00 | Direct Access | 54.00 | Framework III | 389.00 | Language Master | 79.00 | MS Windows v3 |
| C-DOC | 129.00 | Direct/Net | 149.00 | Freedom of Press | 299.00 | LANSpool | 249.00 | MS Word |
| C Asynch Manager | 149.00 | Disk Explorer | 59.00 | Fresh Utilities | 149.00 | Lantastic Starter Kit | 379.00 | MS Word/Windows |
| C Tools Plus | 109.00 | Disk Manager | 69.00 | Fun School III | 21.00 | Laplink III | 89.00 | MS Works |
| Cache86 | 39.00 | Disk Manager-N | 159.00 | Funcky Library | 149.00 | Laser-Ready | 85.00 | MSDOS 3.x w/GWBASIC |
| Calendar Creator Plus | 49.00 | Disk Technician Advanced | 110.00 | GB/Executive | 169.00 | Lattice C | 170.00 | MSDOS 4.x w/GWBASIC |
| Canvas/Mac | 179.00 | Disney Educational | 21.00 | GEM Artline | 389.00 | Le Menu | 53.00 | Multi-Edit Professional |
| Captive | 30.00 | Documentor | 179.00 | Generic 3D Drafting | 329.00 | LetterPerfect | 139.00 | Multi-Lingual Scholar |
| Carbon Copy+ | 99.00 | DoDot/Windows | 129.00 | Generic CADD Level 3 | 329.00 | LimSim | 79.00 | Multimate 4 |
| Carbon Copy+ +Host | 125.00 | DOS Partner | 69.00 | Genifer III | 199.00 | LocoScript/PC | 125.00 | Multimate Advantage II |
| Carbon Copy+ AutoPilot | 75.00 | DosUtils | 65.00 | Glyphix/WordPerfect | 89.00 | Lotus 1-2-3 v2.2 | 259.00 | Multiscope/Windows |
| Cardbox Plus | 225.00 | DoubleDisk | 79.00 | Gofer | 59.00 | Lotus 1-2-3 v3.1 | 339.00 | Netlib Library |
| Certus/PC | 179.00 | DR DOS v5 | 110.00 | GoScript Plus | 235.00 | Lotus 1-2-3/G | 550.00 | NetUtils |
| Charisma/Windows | 299.00 | Dr. Solomon's Toolkit | 49.00 | Grammatik IV (UK) | 95.00 | Lotus 1-2-3 for UNIX/386 | 625.00 | Netware C Interfac |
| CheckIt | 85.00 | Dr. Switch | 79.00 | Grammatik IV/Windows | 69.00 | Lotus Agenda | 278.00 | Neuroshell |
| Clarion Pro Developer | 549.00 | Drafix CAD/Windows | 495.00 | Grapher | 149.00 | Lotus Freelance Plus | 279.00 | New Print Shop |
| Clear Plus/C or dBASE | 129.00 | DrawPerfect | 329.00 | Graphics Link Plus | 139.00 | Lotus Magellan | 99.00 | NewsMaster II |
| Clipper v5 | 325.00 | DS Backup Plus | 72.00 | Grasp | 109.00 | Lotus Symphony | 419.00 | Norton Anti-Virus |

THIS IS A SMALL SAMPLE OF OUR RANGE

## WE'LL *BEAT* ANY NATIONAL

Software Paradise is an independent software reseller representing more than 2000 manufacturers with over 10,000 software products for IBM Personal Computers (and the Apple Macintosh) running DOS, OS/2 and UNIX. We offer the most comprehensive range of PC software in the world and have been supplying both users and programmers with quality products and the service at the lowest prices since 1986. "The Discerning Person's Guide to Software" is the most complete Buyers's Guide to PC software available and is published quarterly by Software Paradise. Call us on (+44) (0222) 887521 or complete the coupon to receive your **FREE** copy.

# FREEPHONE 0800 378·873
## ORDER LINE ONLY

PLACE YOUR ORDERS FREE OF CHARGE FROM ANYWHERE IN THE UK

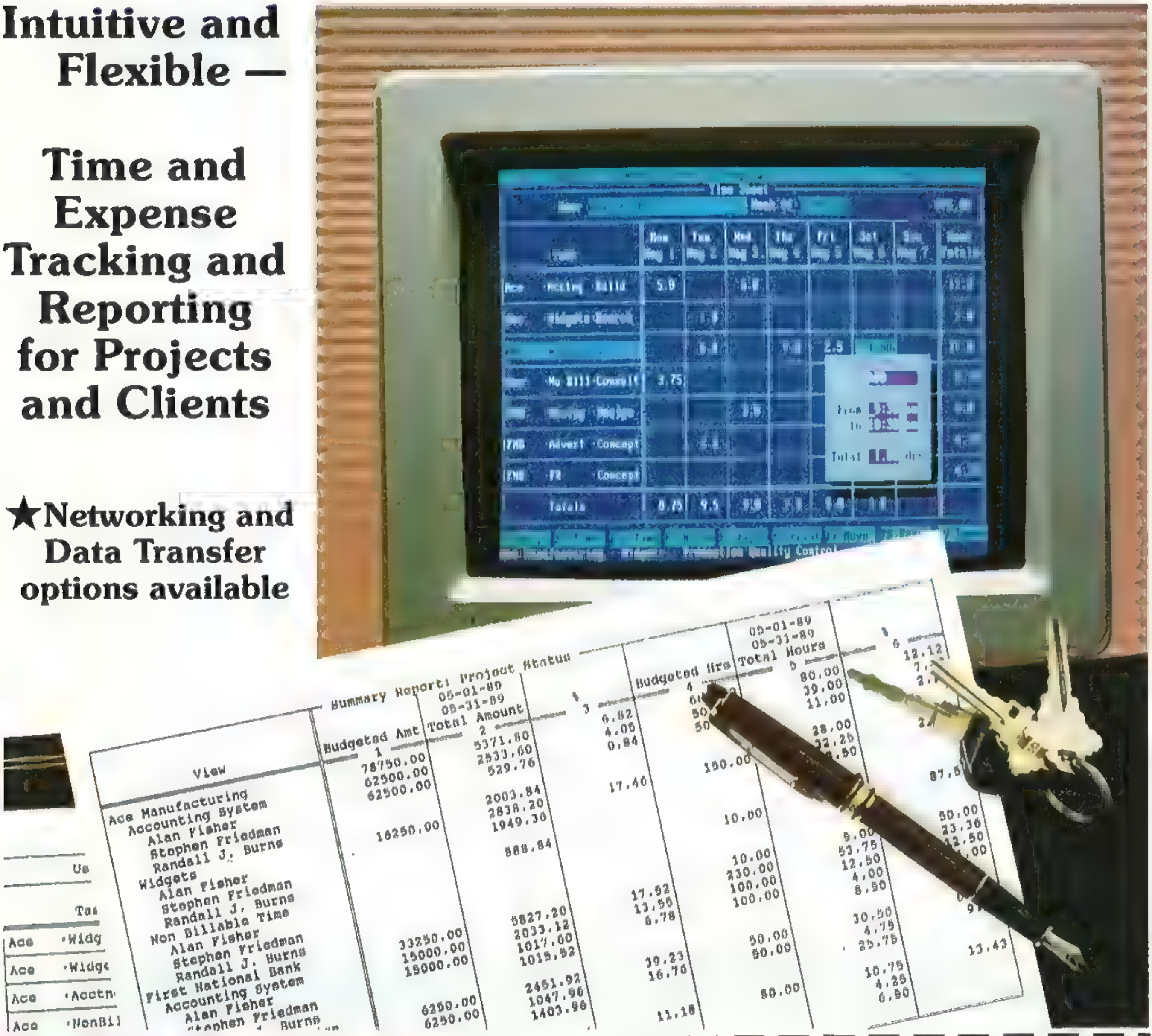
**ALL OTHER ENQUIRIES TEL: (+44) (0222) 887521 (10 LINES) FAX: (+44) (0222) 862209 (24 HRS)**

# TimeSheet™ PROFESSIONAL

Just £99!

**Intuitive and Flexible —**

**Time and Expense Tracking and Reporting for Projects and Clients**



★**Networking and Data Transfer options available**

**DEEPAK SAREEN ASSOCIATES**

**FREEPOST,**
**28 Preston Road,**
**Wembley HA9 8BR.**
**Tel. 081 908 2423**
**Fax 081 904 6668**

☐ Rush me TimeSheet Professional right away at only £99.
Please add £5 handling and VAT. total £119.60

☐ Send me the TimeSheet demo disc plus more information on TimeSheet and the Networking and Data Transfer options.

☐ 5¼" discs ☐ 3½" discs

Name .............................. Company ..........................

Address ..........................................................

..................................................................

Postcode .................... Phone ....................

☐ I enclose a cheque for £119.60

☐ Please bill my Visa/Access/Amex Card

☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Expiry date ........................

Signature ..........................

EXE 2/91

SEND NOW FOR FREE DEMO DISC

CIRCLE NO. 498

# A Video labelled 'X'

*With the price of 'live' training courses, the training video offers an attractive alternative. Willie Watts has been couch potato-ing through an X windows course.*

The *All-Hands-On X Video Workshop* is a new US product comprising 11 videos (each about 20-30 minutes long) in two volumes: *An Introduction to the X Window system* and *Programming with XLIB.* Accompanying documentation consists of a copy of the standard three volume *X Window System User's Guide*, and a workbook consisting of a transcript of the videos with exercises inserted at relevant points. This Christmas, while the rest of the world was snivelling over *ET*, I slogged through the adventures of Bif and Marilyn.

Bif Boston *(sic)* and Marilyn Megabucks *(sic again)* are our hosts for the course. Bif, his slightly dopey face jaundiced by the conversion to the British PAL video standard, is the Comic Relief. Handsome Marilyn, whose Sino-American complexion stands up rather better to this treatment, plays the Love Interest. Sat together behind a desk, as though reading News At Ten, they indulge in a scripted repartee which finally proves, if ever there was any doubt, that there is a gap of about 15 degrees between the British and US sense of humour. (**Marilyn:** 'Hey, Bif, what are you so happy about anyway?' **Bif:** 'I finished my beautiful thing.' **Marilyn:** 'I beg your pardon?' **Bif:** 'My beautiful picture! Take a look.' **Marilyn:** 'A house! What a nice house! It makes the Mona Lisa look like a kid's drawing.' *She looks to camera conspiratorially and rolls her eyes.* 'Or is it the other way around?')


The three videos of Volume 1, the introductory section, not only introduce us to Bif and Marilyn, but contain a mass of in and extra-studio comic business. The concept of the graphics server, for example, is illustrated by an acted out piece set in a burger bar, where the food-orders represent keystrokes, the food is the graphics and there are three kitchens (local, LAN and WAN) illustrating the effects of varying response time. In places the surrealism is cranked up to overload. The X philosophy is explained by, among others, a vacuum cleaner salesman, a Terry Jones style transvestite and a naked man standing in a shower cubicle.

None of the material in Volume 1 is very technical, presumably because it is intended to be palatable to Suits. However, you end up with a very complete picture of what the X components are, and how they all fit together. If you are hazy on the relationship of window manager products like Motif to X itself, then this will sort you.

In Volume 2 we get down to code level. Bif and Marilyn come into their own here, maintaining a constant sexual tension, slightly reminiscent of Steed and Mrs Peel in The Avengers. Whenever the script begins to drag ('For drawing figures that are made of lines, as opposed to filled figures, we have `XDrawLine`, to draw a single line, `XDrawLines`, to draw multiple connected lines, and `XDrawSegments`, to draw multiple non-connected lines. We also have `XDrawRectangle`...') Bif will liven proceedings, perhaps by climbing into the X window he has just coded - a CSO effect - in order to impress Marilyn (no chance: she just does some more eye-rolling). There are also plenty of animated diagrams, a token 'real programmer' called Warren Watcher (but we can tell that he's really an actor; he changes his T-shirt too often) and a truly dire cartoon character called Mr Event Switch... but B and M are the real stars.

The second volume, then, covers all the important XLIB C calls, explaining their function in the context of example programs. It builds up from elementary topics, such as program structure, through to font manipulation, colours (the module is filmed in black and white) and bit images. Note that the material deals with the lowest X API level - no coverage of 'widgets' or 'toolkits' here.

Since I did not have the necessary computer equipment to hand, I missed out on one major component of this course: the 'Hands-on' practicals. But that the videos sustained my interest, more or less, *without* getting to try it all out.

To summarise: although the (intentional) laughs are not up to the standard of an ex-Python or a Smith & Jones production, I enjoyed watching the *X Video Workshop* a lot. And when I'd finished wincing at gauche Bif and squirming at Marilyn's coquettishness, I discovered that I knew rather a lot about X. Product performs to spec. **Marilyn and Bif:** *(waving)* 'Bye for now!'

EXE

---

*X Video Workshop is distributed by MMT UNIX Education (071 278 6211). Prices are as follows: Volume 1 (three videos) £1000, Volume 2 (eight videos) £1700. Documentation sets are priced at £65 and £125 respectively, and the videos are available in both VHS and Beta.*

# Memoirs of a Frontier Programmer

*What's over 30 years old, has a CHAPTER 0 and goes hoot? Ask George Ruscoe, a veritable software old-timer, who has been looking back to his first experiences of a high-level language.*

Recently I was giving training in some EDI software tools to a group of young Graduate Programmers. It came as something of a shock to realise that none of them was born when I first learned to program using Mercury Autocode. Perhaps it was time to begin my memoirs! Here is a start.

Mercury Autocode was first implemented on the Ferranti Mercury in March 1958 by a team lead by Prof R A Brooker, who had also developed the previous Autocode for the Mark I computer. (He subsequently went on to do work on phrase structure and formal language description, and this led to the Compiler Compiler, which was the first successful attempt at a formal language for the transparent description of syntax and semantics.) The program in Figure 1 shows a solution of the familiar Triangle problem. If you have read a previous Third Side article on the original Manchester Autocode (.EXE, April 1990), you will notice many similarities, but also some significant differences.

## Arithmetic

The notation of Mercury Autocode owed a good deal to the conventions of mathematics, so that variables were represented by letters, such as `X` and `Y`, and subscripted variables by `A1` or `AK`. This was a leap forward from the original notation, which only allowed variables called `V1` to `V8192` - effectively one big array. Arrays were declared by the directive `A->10` which set aside 11 elements named `A0` to `A10`. In addition there was a set of *primed* variables, `A'`, `B'` etc, and a special variable π (Pi) initialised to 3.14159265 - the latter reflecting the fact that most programs written for the Mercury machine were mathematical and engineering calculations.

**Several of the ladies were of the opinion that high-level languages would never catch on**

These *main variables* (arrays) and *special variables* (the single letters) were represented by 40-bit words and processed in floating point hardware. There were also 12 *indices*, held in of 10-bits, and represented by the letters I to T, so that only the remaining 14 letters could be used for 'variables'. The use of brackets was very restricted, and there was normally no symbol for multiplication, so by convention `A2` and `AK` both referred to array `A` with a subscript, while `2A` or `KA` implied multiplication. The most complicated term that could be written was `3JKXA(N+5)/Y`, which in modern terms is `3*J*K*X*A[N+5]/Y`. There was often a need to form preliminary sums, especially for a denominator. It was a convention to use certain variables (`W1`, `W2` etc) only for the working intermediate variables; I have used `W` in this way in the example program.

The only use of brackets was for a *complex* subscript, which took the form *index + integer.* An integer 'arithmetical instruction' could only include integers (see Figure 2), but if the left hand side was a real variable, the right hand side could contain either type - an early form of mixed mode expression. The language also provided some standard functions, the common ones being named and preceded by the Greek letter phi, eg `ϕSQRT(X)`. This was an improvement on the F1 notation of the original Autocode, although many of the less common functions were still given numbers; for example `ϕ16` would transpose a matrix and `ϕ8` would print it.

## Input/Output

In general, the old system of abbreviating statement words to one letter had been abandoned. `J` had expanded to `JUMP`; `H` (halt) was now `END` and input was by

```
TITLE
TRIANGLE CHECK - DATA = THREE SIDES

   CHAPTER 0

   READ(A)
   READ(B)
   READ(C)
   W=B+C
   JUMP 1,A≥W
   W=C+A
   JUMP 1,B≥W
   W=A+B
   JUMP 2,W>C
1) CAPTION
THIS IS NOT A TRIANGLE
   END
2) JUMP 3,A≠B
   JUMP 3,A≠C
   CAPTION
THIS IS AN EQUILATERAL TRIANGLE
   JUMP 9
3) JUMP 4,A=B
   JUMP 4,B=C
   JUMP 4,C=A
   CAPTION
THIS IS A SCALENE TRIANGLE
   JUMP 9
4) CAPTION
THIS IS AN ISOSCELES TRIANGLE
9) END
   CLOSE
```

*Figure 1 - Solution to the Triangle problem*

```
A->99                       allow A0 to A99 as an array
I=2MN+M+N+1                 integer
X=2NA1-MA2+5MNA(N+2)/D      float
Y=2π φSIN(X)                standard function
(U,V)=(X,Y)*(A,B)           complex arithmetic
φ6(X)W,N                    transfer N variables from Auxiliary store
                            at X to working store at W
A'=φ11(B',C',U)             matrix A=B+C of size U,were A',B' & C'
                            are locations of matrices in auxiliary store
```

*Figure 2 - Sample Arithmetic Statements*

READ. Output had also been much improved with the PRINT command (see Figure 3) which would take any variable expression X and convert it for output with M characters before the point and N after it, suitably rounded. Numbers would always be followed by two spaces, but additional control could be gained by SPACE and NEWLINE instructions. On input, a number was terminated by two spaces or carriage return and linefeed, so that input and output were compatible. Remember that in this era, input was prepared by punching a five channel tape on a teleprinter, and the output was produced on a punched tape which was then printed out offline.

There was a facility for printing the result of any calculation by placing a question mark at the start of the statement. This was regarded as a form of diagnostic print (called a query print) and could be switched on or off by setting hand switch four on the console.

In the example program, the program tape would be read in and the program compiled. When CLOSE of CHAPTER 0 was encountered, execution would start at the first instruction of that chapter. At this point, the computer would try to read in a data tape which you should by now have loaded. Two features shown in the program were later additions. The TITLE directive was introduced so that the output could be identified! This was acted upon as the tape was read in, and the contents of the following line were copied to the output punch. CAPTION was a similar run-time instruction which caused the content of the following line to be output at that point in the program. Prior to this, your output would consist of a set of numbers only, unless you inserted machine code instructions in your program to output a character at a time.

## Organisation and Control

Mercury Autocode was conceptually divided into four areas: Arithmetic & Storage, Input/Output, Program Organisation and Control. The first two groups we have already seen. The simple Control instructions of unconditional and conditional jumps are much the same as in the original Autocode. There were symbols for 'greater than', 'greater or equal', 'equal' and 'not equal', and you could compare two variables or a variable and constant. Note that if you wanted to test for 'less than' you had to reverse the condition and jump the other way! Labels were separated from the statements by a right bracket (rather than the colon of modern languages).

***A table gives the time taken up by each instruction and operator - in micro minutes***

One of the major new features was the cycle (loop). This consisted of a pair of instructions: I=N(S)M and REPEAT (shown in Figure 4) which came at the start and end of the loop. The step S could be either positive or negative, and the items were all indices or constant integers. You had to make sure that M-N was a multiple of S, as the terminating test was for equality. Loops might be nested to a depth of eight and were one of the first high-level control instructions. Another interesting control statement was the special form of test, CHECK, which had the effect 'jump to label 3 if E is greater than the absolute value of (X-Y)'. This was used to test for convergence.

The other important control feature - the subroutine - was implemented in a number of ways. The design of the first was more or less determined by the limited size of the machine. The main store (which was made of magnetic cores) was limited to 1024 words of 40-bits each. Backing store was held on magnetic drums. Half of the main store could contain your program, the rest held data. Autocode programs were restricted to 832 instruction registers (these were medium sized 20-bit words) and the main variables were restricted to 480. You could declare A->479, but that was your lot.

So far as the program code was concerned, any large program was split into chapters, headed CHAPTER 1 and so on, each of which had its fixed place on the drum. The space not used by these was available for auxiliary variables up to *(10,752 - 512N)*, given *N* chapters. As they were compiled, these chapters were stored away, but when Chapter 0 arrived (at the end), it was compiled and put into execution, starting at the first instruction. A small program might be held entirely in this chapter, but usually it was necessary to split. You transferred control to another chapter with ACROSS 1/2, which meant execute chapter 2 starting at label 1. Thus multiple entry points were provided by different labels.

To treat a chapter as a subroutine you could use DOWN 1/3 and return to the calling chapter with UP. If you wanted to look after your main memory variables during the call, you could use PRESERVE before DOWN and RESTORE immediately after. This would guarantee that the subchapter did not disrupt the variables in the calling chapter, although any values set before the DOWN were still available in the sub-chapter. To return values, you have to place them in the auxiliary variables on the drum before the RESTORE took effect.

It was very inefficient and slow to swap chapters every time a small routine was called. This problem also dogged standard library functions. Although some of these were coded inline, most were pulled in from the drum when needed. If you had room in your chapter, you could list the

```
READ(I)
READ(X)
PRINT(X)M,N
SPACE
NEWLINE
HALT                        pause with hoot, wait for key
HOOT n                      n=1 to 8 in the major scale
CAPTION
LINE OF CAPTION...
```

*Figure 3 - Sample Input/Output Statements*

```
I=1(1)N                                              start loop
 PRINT (AI)5,2
REPEAT                                                 end loop
JUMP 9, I=N                                         conditional
CHECK (X, Y, E, 3)                            test difference X-Y
END                                                        stop

L)=3)                L becomes address of label 3 JUMP 20
                                                 to subroutine
3)...                               return label (for this call)

20)...                                           the subroutine
JUMP L)                                  return from subroutine

L)=N)                         if N=3, L is address of label 3

JUMPDOWN 50                                    subroutine entry
RETURN

0)                             Label to trap READ fault (32)
100)                       Label to continue on other faults
```

*Figure 4 - Sample Control Statements*

functions in order of priority, and those which would fit - known as 'quickies' - were incorporated.

It was possible to include a small subroutine in a chapter, but you had to manage the call and return yourself. It was possible to store a label value in an index, say `L)=3)`, before entering the routine with a `JUMP`. The return was achieved with a `JUMP (L)`. Eventually a new instruction, `JUMPDOWN`, was added. This noted its own address on the stack, and was paired with `RETURN`, which worked as you would guess. Parameter passing, however, never progressed beyond setting global variables as detailed in the specification of the routine.

## Further Features

The later versions of Mercury Autocode were called 'CHLF Autocode', named after the sites at Cern, Harwell, London and Farnborough where the various enhancements had been developed. In the final release, CHLF3, a form of internal procedure had been invented. This was declared like this `ROUTINE 50` and entered by `JUMPDOWN(R50)` or `JUMPDOWN(R50/3)`. It boasted its own set of internal labels, distinct from those of the main routine - but still shared the same variables.

Other advanced features offered by the language included random, complex and double precision numbers, matrix manipulations (all matrices were held on the drum) and step by step integration of differential equations by the `INT STEP` instruction. A considerable library of programs and routines was built up, mainly of a mathematical nature.

There was a **lot** of interest in the size and speed of programs. For instance, there is an instruction `N)=M)` for setting a switch label N from an integer value M, but the manual tells us that 'the execution time of *n)=m)* is very much longer (18 millisecs) than that of *n)=3)* (120 microseconds).' There is also a table giving the space and time (in micro minutes) taken up by each instruction and operator! The main use of this was to emphasise that a divide was 12 times as slow as a multiply and 60 times as slow as an add, while chapter changing and input/output in general were very time consuming (as one might expect). However, it was often a matter of shoe-horning a program into the least space, and trying to optimise speed, so this information was valuable.

```
ACROSS 1/2                     to label 1 chapter 2

CHAPTER 2
1)...
PRESERVE
DOWN 1/3
RESTORE
...
CLOSE                          textual end of chapter

CHAPTER 3
1)...
...
UP
CLOSE
```

*Figure 5 - Sample Organisation Statements*

## Female Luddites

When I first met Mercury Autocode in 1962, it was already well established, although the CHLF3 version had yet to appear. However, several of the ladies with whom I worked (they were in a majority in the department) were of the opinion that these 'high-level languages' would never catch on with professionals. They continued to code away busily in purest assembler. But FORTRAN was already available, and soon we were beginning to use ALGOL on the English Electric KDF9.

Meanwhile one of my colleagues came back from the USA with tidings of a new language. It had variables called `A1`, `X9` and `READ` and `PRINT` statements; `GOTO` and `IF` jumps, `FOR` and `NEXT` loops and `GOSUB` and `RETURN`. At first sight it didn't seem to have much over Autocode, apart from more general arithmetic expressions. What I didn't realise at the time was that you used it directly from a time sharing terminal. It was called BASIC, and by early 1968 I found myself teaching it in half day courses every Wednesday morning.

Since then, I have spent a lot of time wrestling with BASIC and FORTRAN and PL/1 and C, but I still look fondly back to the days when you took your paper tape down to the machine room and fed it into the beast yourself. Of course, the time sharing system brought immediacy to the user, and the advent of the PC has put far more power on the desk that ever we had in that huge machine room. But somehow it's not quite as impressive, and anyway with modern computers without refrigeration units, there is nowhere to keep the milk cool. Ah, the price of progress!

EXE

*George Ruscoe began his programming career at AEI in Manchester. In more recent times he has worked from home with a new-fangled PC, and is currently employed by software consultants Humphreys & Dean.*

C++ DEVELOPER'S EDITION V2.1


# For total portability right now, your choices are a bit limited.

## Only Zortech Gives You Total Portability to MS-DOS, Windows, OS/2, DOS 386, UNIX 386. And Macintosh. Right Now.

### What platform will your application be running on this time next year?

Now that C++ has made significant inroads into mainstream application development, you must choose the C++ compiler that is going to be the performance and portability leader now...and years to come. Since 1986, we've been testing, refining and perfecting our family of C++ products tirelessly. The results are what you are reading about, a fully portable family of C and C++ products available right now.

BUYERS ASSURANCE SEAL INFO WORLD

### We Intend To Keep Up This Pace.

In the past few years, Zortech has quietly become an industry standard earning acclaim from Byte, Computer Language, Dr. Dobbs, and many industry experts for our solid implementation, market-leading performance, and strong, relentless innovation.

### Whatever Platform You Choose, Choose Zortech Performance.

With Zortech, no matter what platform you choose to develop on, a substantial performance increase will be realized due to the quality of the original Zortech C++ implementation. For instance, on MS-DOS, your program will compile up to 35% faster with the resultant code running up to 45% faster and 25% smaller than Turbo C++*. But we don't stop there. On each platform, Zortech C++ is designed to improve your productivity where it really matters... in the development cycle.

MICROSOFT WINDOWS

For example, using the DOS/MS Windows package, you can edit, compile, link and debug the largest MS windows applications...and never once leave the Windows environment. All this plus complete plug-and-go to CNS C++/View and Commonview II! And, of course, these advantages are also available on the OS/2, DOS 386, UNIX, and Macintosh platforms.

60 DAY MONEY-BACK GUARANTEE

### Choices, Choices, Choices...

If there is one constant in C++ development, it's change. The Zortech family of products keeps your options exactly how they should be. Open.

To find out more, call today.

ORDER HOTLINE:

44-81 316-7777

# Zortech

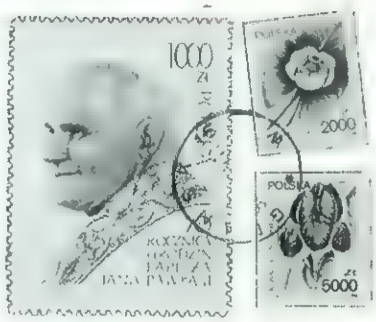
Industry Standard. Industrial Strength.

ZORTECH, INC., 4-C GILL STREET, WOBURN, MA 01801 TEL: (617) 937-0696 FAX: (617) 937-0793
ZORTECH, LTD., 58-60 BERESFORD STREET, LONDON SE18 6BG TEL: +44-81-316-7777 FAX: +44-81-316-4138
SUPPORTED SYSTEMS INCLUDE COMPAQ, ISA/EISA, IBM PS/2 AND 100% COMPATIBLES

*Source: MS-DOS C++ Compilers, A comparison by the Ladd Group, (c) 1990.

CIRCLE NO. 199

# The Polish Connection



*A new year, and a new .EXE regular column, which will look at software development issues from a foreign point of view. Richard Sadowski kicks us off with a Polish perspective.*

Poland has a long and honourable tradition of computer science. The father of computer calculation was the Polish scientist Lukasiewicz, probably best known for so-called Reverse Polish Notation. Another claim to fame is that a Pole, a Mr Karpinski, invented one of the first minicomputers 20 years ago. But Karpinski's project was abandoned because of the political factors: it didn't fit into the Eastern (= Soviet) standards.

Karpinski was forced to forget his dreams and to work as a farmer for many years - but the story has a happier ending. With the new wave of the Solidarity movement he was able to return to his original interest. Now he runs his own company abroad and is a respectable engineer and manager. His story is very typical of our older computer engineers and programmers.

Before the era of PCs, the only machines used in Poland were the ODRA (a Polish version of ICL 1900), RIAD (a Russian IBM 360-alike mainframe), SM-60 and MERA (both based on DEC's PDP-11). These were situated in the academic institutions and large Government computer centres. At that time, the demand for computer science graduates was slack. There were no economic reasons to change this situation, as the jobs market was controlled by a few state-owned companies.

The first wave of the computer 'madness' arrived in Poland in 1985. Almost every technical magazine printed a short course on BASIC. Everyone wanted to become a programmer. The excitement only calmed down when people started to understand that it was impossible to write all the software they needed from scratch.

Poland was flooded with hundreds of home and personal computers. The most important machine of those times was the ZX Spectrum which, thanks to its low price and the fact that it could be exported from the West without a licence, enjoyed tremendous popularity not only among the game enthusiasts but also in the serious scientific institutions and state companies. For people used to fighting with card punch machines, even this was a real step forward.

***The shortage of software was overcome by not introducing any software copyright laws***

Fortunately the Spectrum soon began to give way to more serious personal computers. Again, price and COCOM restrictions dictated choice of machine. Ninety per cent of the Polish market was cornered by cheap Taiwanese PC clones.

However, computers were not yet viable for everyday applications. There were two reasons for this. The first was cost: the price of an average PC was approximately equivalent to a programmer's wages for 10 years. The second problem was lack of software. It took about five years to reduce significantly the price of computers - they are now worth about six months' salary. The shortage-of-software obstacle was simply overcome by the old Communist government, which chose not to introduce any software copyright laws.

All kinds of software were copied freely. This situation had some advantages. Few companies and developers could afford to pay the full price for the software, as Western software houses didn't discount their products when selling in Poland. However, there were big drawbacks as well. Polish software developers could not sell their products. Anything that reached the market was instantly pirated. We hope that this situation will change soon, when our Parliament sets a new copyright law. For the time being, most of the programs sold in Poland are protected by various software and hardware keys.

In 1990, with the easing of COCOM restrictions, a second wave of computer equipment started to come to Poland. Even though PCs continue to dominate, an ever greater variety of systems is penetrating the Polish market. There are VAXs, Unisys machines, ICLs, IBMs and others. Lack of software to run on them remains a major problem. We have a large number of a talented programmers, but most applications still have to be written from scratch. There are some attempts to customise Western products, but until Polish regulations become more 'European' it will remain a Sisyphean task.

Recently, some Polish software companies have entered into partnerships with Western developers. This kind of cooperation should bring mutual benefits. First, it can help to solve the problems with the lack of manpower in some Western countries. Second, it may cut down the development costs up to five times. Finally, it may help to decrease the technological gap between West and East.

EXE

*Richard Sadowski has run a private computer company 'Studio Uslug Komputerowych SAMBA' since 1986. The company is both a computer dealer and software house. Notable products include a suite of programs for civil and mechanical engineering, and it is working on various business applications and CASE products intended for a wider market. The author can be reached on 010 4858 217088 or fax 010 4858 204940.*

# Two Column Listings

*Dan O'Brien presents a plan to save the rain-forests via an undocumented feature of some Epson compatible printers.*

It has always mildly annoyed me that printed program listings never quite reach the end of the line. Like poetry - terribly wasteful and namby-pamby. You'd never catch a hex dump missing valuable space like that. Eighty-column documentation grates, too, when it's sprawled across an A4 size page. So, in a spirit of recession-combatting, tighten-our-belts eco-thriftiness, this month's Code Page presents a short C program to print out programs and documentation in two A5 columns of 80 characters each.

According to dogma, this is impossible on strictly Epson compatible printers, which can only run 132 columns in condensed pica mode. Fortunately, what is impossible in theory is entirely reasonable in practice, as 160 columns is generally an easily added feature for compatible manufacturers to implement. The actual control codes vary between brands; the most common is to use the Epson condensed text code (ASCII 15) while in elite mode (activated with ESC <M>). These are the codes used here. Users of other printers can discover their own 160 column mode commands through a cursory manual browse.

> ***The program sacrifices speed for readability, I'm afraid***

The program itself is a cinch. As you would expect, it uses a buffer to read in text a page at a time, rearranging the text as it prints. The toughest code is reading in the command options. It should run on any ANSI C system, although you may have to implement your own version of `strdup` if your libraries do not supply it. Watch out for the 'rt' option in the input file `fopen` - that 't' isn't ANSI, but many PC compilers need the

```
/* PR2- Two column printer */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#define TRUE 1
#define FALSE 0
typedef int BOOL;
/* Procedures:    */
void setupPrinter();/*initialise output stream */
void printOut(char *);/*print single file*/
BOOL loadColumn(int,FILE *);/*read one column */
void expandTabs(char *);
void overflow(char *); /*trap overlong lines*/
void printPage(void);  /*print full page*/
char *skpspace(char *);/*skip  white space*/
void xerr(char *,int); /*error handler*/
/* Non-ANSI functions:  */
char *strdup(char *c); /* gets malloced copy*/

#define MAXLINELEN 1024
#define MAXPAGESIZE 100
#define CWIDTH 80       /* Width of column    */
/* Printer codes /
char *prFF="\014";
char *prSeparator="\10|";
char *prSetup="\x1BM\x0F";
/* An alternative code is \x1B!\x05   */
/* default values for all the options...*/
BOOL separate=FALSE;
int tabSize=8;
char *prFile="LPT1:";
int pageSize=60;
char page[2][MAXPAGESIZE][CWIDTH+1];
/* emptyStr should have CWIDTH spaces in it */
char *emptyStr="                                        \
                                        ";

FILE *printer;
int main(int argc, char *argv[])
{
char *optarg;
int optind=1;
if (argc==1)
  |
  xerr(" \
    Usage:pr2 [-s] [-o <outfile>] [-t <tabsize>]
[-l <length of page>]\
  <fname>\n"
  ,EXIT_SUCCESS);
  }
```

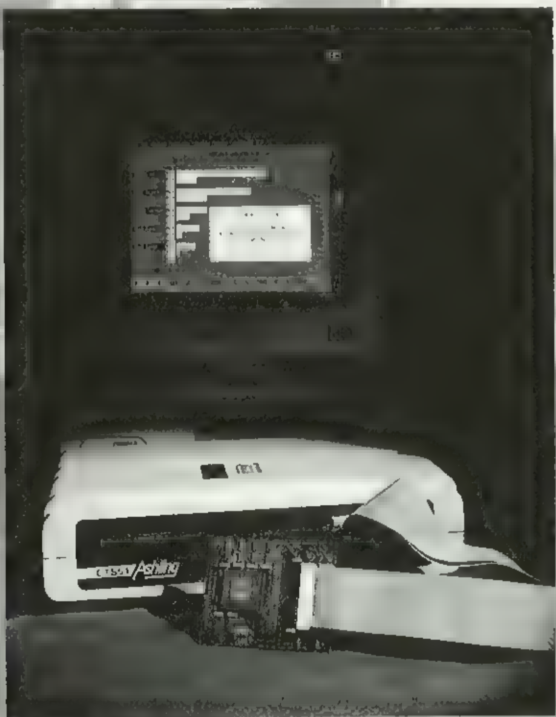
*Figure 1 - The Pr2.c listing*

# 8051
# 68HC11

**CT SERIES MICROCONTROLLER DEVELOPMENT SYSTEMS**
for 8051 and 68HC11 embedded systems designers

Ashling Microsystems has set the standard for high-quality Microcontroller Development Systems since 1984 and continues to provide the outstanding package of features. In particular, Ashling's latest CT-Series products provide you with unique capabilities that will help you deliver on your design project targets.

**Management of Variables**: The Ashling "PathFinder" High Level Language Debugger can interpret the most complex array and structure variables. Out-of-bounds array indices are detected and flagged. The Variables Monitoring Window is updated to reflect the current contents of the simple type, the specific complex array element, or the individual structure member referenced. Expressions involving variables of simple and complex type can be formed and are updated in the variables window.



**Automatic Synchronisation of Debug Windows**: PathFinder links your Software and Hardware painlessly and efficiently by automatically synchronising the following windows; Original Source Code; Disassembled Memory; Traced Source Code; Traced Bus Activity; and Traced I/O Port Activity.

**System Performance Verification**: The CT series can provide a continuous, real-time measurement and log of system responses and can verify that you have tested for and met the performance specifications of your design. You can determine (to 2 microsecond accuracy): interrupt service times; frequency of interrupt occurrence; elapsed time between interrupts; system execution time distribution; and timestamped function flow trace.

**Efficient Operating Environment**: You can move quickly from one development, debug, test and validate function to the other. The Ashling Development Environment gives you, Software Development; High-Level Language Debug; In-circuit Emulation; Real-Time Trace; System Performance Verification; PROM Programming; On-Line cross referenced HELP; and External File Browse; all seamlessly integrated in a single debug environment.

**Cost Effectiveness**: The CT-series acquisition options are also flexible and customer requirement orientated. System prices start at £ 2,999 with a range of modular, inexpensive upgrade paths. There are Rental, Rental-to-Purchase, Purchase and Lease Plans; while Renewable Warranty and Software Update Plans provide control of maintenance costs and protection of your investment into the future.

**See us on Stand No 26**
**MDS, Wembley Conference Centre**
**Feb 5th-7th 1991.**

**Telephone or fax *now* for Interactive Demo Disk, On-site Demo or further information:**

Ashling
The Development Systems Company

**Ashling Microsystems Limited**
**Butler House**
**19-23 Market Street**
**Maidenhead, Berkshire,**
**SL6 8AA U.K.**
**Tel: (0628) 773070 Fax: (0628) 773009**

CIRCLE NO. 503

text mode option to be made explicit, as it involves some data translation.

The form of the program sacrifices speed for readability, I'm afraid. The data structure for text, for instance, should make it easy to add more columns. This is a working program in the sense that I've been using it for some years, so there are a couple of 'features' which I couldn't bring myself to rewrite. The most glaring is the static declaration of the buffer arrays, and the huge string allocations. I instinctively shy clear of `malloc`, and when I do use it, tend to snatch huge lumps of KB and then hoard it. In an ideal world, this wouldn't be necessary; at present, I have more memory than I do debugging patience. Another kludge is the hard-wired nature of some of the options, notably the printer control codes. Again, this is due to a personal abhorrence of huge option lists in command lines, and a financial tendency to have only one printer. Finally, PR2 should, I suppose, wrap-around overly long text. Indeed, it used to, but I found this more of a hindrance than a help in listings. A low tab value is a better solution.

The normal glowing praise for my indentation style: quite the clearest and most useful I've ever seen. Although someone did point out that if I really was so keen about saving the rain-forests, I wouldn't give those right-hand brackets their own lines. Well, what's life without a little extravagance?

EXE

*Dan O'Brien is .EXE's News Editor. He can currently be found in section 12-24 of the Windows Programmer's Reference, lost and very alone.*

```
for(;*(optarg=skpspace(argv[optind]))=='-';
optind++)
  {
  switch (*++optarg)
    {
    case 's':
        separate=TRUE;
        break;
    case 'o':
        if (++optind!=argc)
          prFile=strdup(argv[optind]);
        else xerr("What output file?\n",
          EXIT_FAILURE);
        break;
    case 't':
        if (++optind!=argc)
          tabSize=atoi(argv[optind]);
        else  xerr("What tab size?\n",
          EXIT_FAILURE);
        if ((tabSize<0)||(tabSize>CWIDTH))
          xerr("Silly tabsize\n",
          EXIT_FAILURE);
        break;
    case 'l':
        if (++optind!=argc)
          pageSize=atoi(argv[optind]);
        else  xerr("What page length?\n",
          EXIT_FAILURE);
        if (pageSize>MAXPAGESIZE)
          xerr("Page size too big!\n",
          EXIT_FAILURE);
      break;
    default:  xerr("Bad option\n",EXIT_FAILURE);
    }
  };
setupPrinter();
if (optind==argc)
 xerr("What input file?\n",EXIT FAILURE);
while (optind<argc)
  printOut(argv[optind++]);
fclose(printer);
return(EXIT_SUCCESS);
}
char *skpspace(char *i)
{
while ((isspace(*i)) && (*i)) i++;
return i;
}

void setupPrinter(void)
{
printer=fopen(prFile,"wb");
if (printer==NULL)
  xerr("Can't open printer for output\n",
      EXIT_FAILURE);
fprintf(printer,prSetup);
};

void printOut(char *fname)
{
FILE *inFile;
BOOL atEnd;
inFile=fopen(fname,"rt");
if (inFile==NULL)
  xerr("Error: Couldn't open input file\n",
EXIT_FAILURE);
atEnd=feof(inFile);
while (!atEnd)
  {
  atEnd=loadColumn(0,inFile);
  if (!atEnd) atEnd=loadColumn(1,inFile);
  printPage();
  };
fclose(inFile);
}

BOOL loadColumn(int pageNum,FILE *inFile)
{
char (*currPage)[MAXPAGESIZE][CWIDTH+1];
BOOL fin=FALSE;
int lineNum=0;
currPage=&page[pageNum];
for(;((lineNum<pageSize)&&!fin);lineNum++)
  {
  char currLine[MAXLINELEN];
  int currLen;
  char *tmpPnt;
  if (fgets(currLine,MAXLINELEN,inFile)==NULL)
   fin=TRUE;
  else
    {
    expandTabs(currLine);
    if ((tmpPnt=strchr(currLine,'\n'))!=NULL)
     *tmpPnt='\0';
```

*Figure 1 - The Pr2.c listing (continued)*

CIRCLE NO. 504

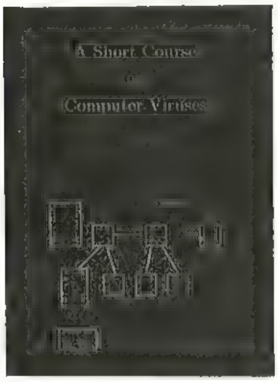
```
    if (strlen(currLine)>=CWIDTH)
    overflow(currLine);
    if (strchr(currLine,'\14')!=NULL) break;
    strncpy((*currPage)[lineNum],
  currLine,CWIDTH+1);
    };
  };
return fin;
}

void expandTabs(char *currLine)
{
char tmpLine[MAXLINELEN];
char *tmpPnt=tmpLine;
char *currPnt=currLine;
while (*currPnt)
  {
  if (*currPnt=='\t')
    {
    int sizeOfTab;
    sizeOfTab=tabSize-((tmpPnt-tmpLine)%tabSize);
    for(;(sizeOfTab>0);sizeOfTab--)*tmpPnt++=' ';
    currPnt++;
    }
  else *tmpPnt++=*currPnt++;
  };
*tmpPnt=0;
strcpy(currLine,tmpLine);
};
void overflow(char line[])
{
fprintf(stderr,"Line too long!\n%s\n",line);
fprintf(stderr,"Chopping to %d chars \n",CWIDTH);
line[CWIDTH]='\0';
};

void printPage(void)
{
int lineNum;
for(lineNum=0;lineNum<pageSize;lineNum++)
  {
  strncat(page[0][lineNum],emptyStr,
          (CWIDTH-strlen(page[0][lineNum])));
  strncat(page[1][lineNum],emptyStr,
          (CWIDTH-strlen(page[1][lineNum])));
  fprintf(printer,"%s",page[0][lineNum]);
  if (separate)
    fprintf(printer,"%s",prSeparator);
  fprintf(printer,"%s\n",page[1][lineNum]);
  page[0][lineNum][0]='\0';
  page[1][lineNum][0]='\0';
  };
fprintf(printer,prFF);
}

void xerr(char *s, int r)
{
fprintf(stderr,s);
exit(r);
};
```

*Figure 1 - The Pr2.c listing (continued)*

# Books

*Tomes that look at viruses and data structures.*

## See It, Kill It, Get Away Quickly




Dr Frederick B Cohen has been writing and speaking on viruses since they were just glimmerings in the eyes of perverse minded computer scientists. And, as a perverse minded computer scientist himself, Dr Cohen does know his stuff. He was responsible for one of the earliest academic experiments in virus construction, a 200 line UNIX C program written in 1983, which took out the university computer in five minutes. Cohen was, in his own words, 'somewhat surprised' at the success of the program and has since spent his time trying to convince others about the dangers of viral attack. Not many people listened to Cohen in 1983, but they certainly do now. *A Short Course In Computer Viruses* is Cohen's way of reminding us that He Told Us This Would Happen.

The book itself is a distillate of a one day course Cohen holds and is consequently technical in content while chummy in style. Longer jokes Cohen has moved for continuity's sake to a Funny Appendix; anecdotes are mostly taken from Cohen's Wilderness Years. As far as technical details are concerned, *A Short Course* takes the reader from a brief outline of the theory of viral infection, and from this goes on to demonstrate exhaustively how no computer system can be entirely protected from viral attack. Cohen then demonstrates that most computer systems (even supposedly high security installations, like the Bell-LaPadula security system) are not only vulnerable to viral infection - they are practically begging for it. In Bell-LaPadula, for example, the rule is that you can't read information higher than your security level, and you can't write (leak) information to humbler security levels. It was very popular with the US National Security Agency for a time. Unfortunately, people are nosy, particularly security agency-type boss-people, and (as Cohen points out) access to lower levels is just an invitation to run underling programs, including, of course, any viral programs sneaked in by the mysterious temp with the central European accent.

After dealing with these systems, it's a fairly simple job for Cohen to document the disasters so far, which he does with some glee. As well as high security systems, he also gives some background information to many of the more infamous infestations of the last few years. Also listed are a few theoretical attacks : interesting virus types that (probably) don't exist, but are at least possible. Included are intriguing descriptions of a spreadsheet macro virus, and a very odd tale about Ken Thompson, the man who co-wrote UNIX. Not wishing to pirate all of Cohen's best stories, I won't give away the details, but if Ken wasn't joking, UNIX sysops world-wide can rest easy in the knowledge that at least one of their account-holders knows how to work vi.

Cohen is ultimately cynical about any form of protection, but this does not stop him from going into a detailed examination of current approaches. Part two of the book is devoted to devising practical defences. Technical methods - virus protection programs - he divides into three categories: sound defences that will always limit viruses (though usually with unpleasant side-effects), solid defences which will usually work, given some care, and rubbish defences, which belong in the bin. Interestingly, the one defence which Cohen has no time for at all is virus scanning programs, which he dismisses with some well aimed swipes.

The chapter on non-technical defences takes a broad interpretation of the word 'defence', including as it does techniques for recovering from viral attack (three sections called See it, Kill it, and Get Away Quickly) and how to trace the miscreant.

Cohen will still occasionally mention how it won't do any good anyway, and how we are all doomed. But he does know when to tone down the heavy kismet, and become more practical: chapters 8 and 9 explain how to perform a cost analysis of viral defences, together with a list of approaches for small business, university, financial institutions and computer company installations. Finally, to wrap it all up, there is an annotated bibliography of the subject, including most of the landmark papers.

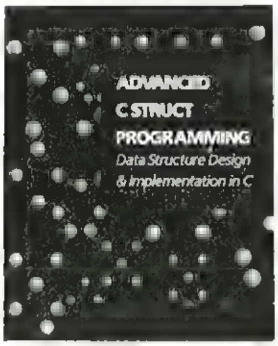
It seems pretty conclusive, and is certainly an entertaining read if you enjoy laughing out loud at the poor misfortunes of others, as I do. Provisos? Well, Cohen does have a few axes to grind, and his belittling of other anti-virus utilities might have something to do with his academic stance as promoter of Integrity Shells (which are, by contrast, heavily plugged as a defence). The mathematics which Cohen occasionally falls back on isn't always necessary, and is occasionally used, it seemed to me, just to make it all seem terribly definitive and clever when it's not. Also, the book is only 200 pages of thinly-spaced text, which is a little dinky for twenty five quid. Still, Cohen does have clout, and is not as self-appointed a viral expert as most. If you're considering how best to defend yourself from the viral lurgy, you could do considerably worse.

*Title: A Short Course on Computer Viruses* *Price: £25.00*
*Author: Dr Frederick B Cohen*
*Publisher: ASP Press (In UK, contact IDS on 071 631 0548)*
*ISBN: 1-878109-01-4* *Pages: 200*

### Books Received This Month

| | | | | |
|---|---|---|---|---|
| *Hypertext and Hypermedia: Theory and Applications* by Nigel Woodhead | Addison Wesley | £19.95 | ISBN 0-20-154442-3 | pp400 |
| *C for FORTRAN Programmers* by T.D. Brown Jnr | Prentice-Hall | £17.95 | ISBN 0-13-120528-5 | pp206 |
| *The Programmers Guide to DB2* by Paul H. Byeall | John Wiley | £24.50 | ISBN 0-471-52227-9 | pp240 |
| *Object-Oriented Languages, Systems and Applications* by Blair, Gallagher, Hutchinson, Shepherd | Pitman | £24.95 | ISBN 0-273-03132-5 | pp370 |
| *The Practical Guide for People With A New Idea* by Laurence Shaw | Patent Eye | £9.95 | ISBN 0-950-8559-01X | pp150 |

## Oceans, webs, spells and structs




The big mystery about *Advanced C Struct Programming* is the title. As its author, John W L Ogilvie is quick to explain, this is not a book concerned with just structs. It's not even just about C. Instead, it's an attempt to discover a structure construction algorithm: a technique, which, if routinely followed, should establish all of the requirements of a data structure before its implementation.

It's a tall order. Ogilvie thinks he knows the answer: a huge checklist of about 50 complicated queries, which, like those quizes in Reader's Digest, reveals all about your data's personality. Seem simplistic? Well, it's only the core of this hefty book, and the questions themselves are not exactly multiple-choice. Question one, for example, asks 'What is the geometry of this abstract data type?', which takes some explaining in itself. You begin to grasp the questions only after ploughing through the many worked examples.

This struct construction kit is part of a wider plan which includes almost the entire life-cycle of a program, from initial specification to debugging and maintenance. Ogilvie has questionnaires for these tasks, too. But even that's not enough; there are chapters on all kinds of levels, from specific debugging tips (including a lovely technique for making dumps readable by choosing hex values which spell out their purpose: `0xDEADC0DE` for example) right through to a very brave attempt to explain how programmers think to the managerial mind. Struggling somewhat here, Ogilvie attempts to convey the act of programming through comparison to swimming in oceans, clutching at webs and weaving spells. The metaphors sound trite, but Ogilvie's descriptions ring true. Sadly, he's trying very hard to do the nigh impossible.

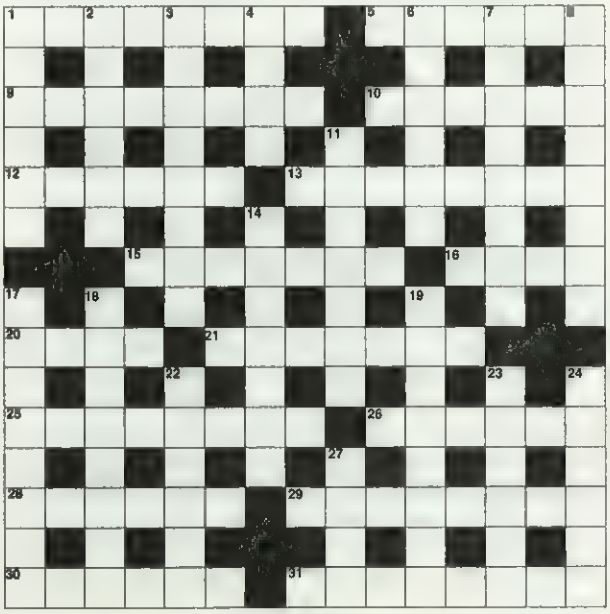
Which rather sums up the book. One can see what Ogilvie is trying to present. Most programmers feel the frustration of being caught in an otherwise precise discipline with nothing but a few rules of thumb; no complete theory, and without any consistently successful approach. *C Structs* is a sort of core dump of practical knowledge, tied to an imposed structure. Ogilvie clearly thinks that his approach to programming is reflected in his set procedure. If you follow the rules, you won't go wrong. In fact, his examples show that he is often hard pressed to squeeze the instinctive approaches he does adopt, to the rigid rules he is claiming to follow.

Ogilvie is obviously an excellent programmer with a gift for communicating concepts. What he tries to do in *C Struct* is write a book that will show everyone how to program so well. Thinking like a programmer, he's convinced that if he can consistently code well, he must be following a set of rules, which he can, in turn, explain to others. This book isn't quite that explanation, but it is very near, and it's certainly one of most complete descriptions of practical programming I've read for some time. A difficult book to recommend, but an author who should be watched out for.

*Title: Advanced C Struct Programming* — *Price: £22.95*
*Author: John W L Ogilvie* — *Publisher: John Wiley*
*ISBN: 0-471-51943-X* — *Pages: 410*

EXE

## .EXEWORD FEBRUARY

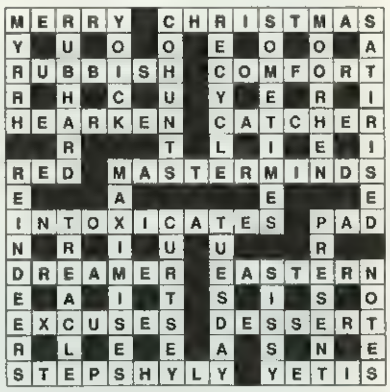


### ACROSS

1 Buses for thieves in the night? (8)
5 Used a 15 to step through the software (6)
9 Hit rate of sporting types (8)
10 Crudely thrash in scalar ruptures (6)
12 Maybe deal with multiples of 2 in a fair way (6)
13 Working eight bits at a time (8)
15 Useful little routine! (7)
16 Spill gore when you meet him in the dungeon (4)
20 Meat to software always... (4)
21 ... and label every chunk of it? (7)
25 & 26 Maybe it has a Centronics port (8, 6)
28 Young doctor may arrive when he should (6)
29 Precise splatterer when painting (8)
30 Common printers (micros?) (6)
31 Goes through it all again conservatively (8)

### DOWN

1 Unusual soccer shot on file (6)
2 Collect 20 or rosebuds (6)
3 Small analog signals produced by veal stew (8)
4 Mysterious 16 of the Himalayas (4)
6 1 down loses head for king in the drive (6)
7 Bearing of printer getting RETURNed (8)
8 Poor layer in a diode (8)
11 Hardware, software - the lot (7)
14 One in a conspiracy may provide the picture (7)
17 Smartly flexible in packet routing (8)
18 31 the routine (8)
19 Its object is to put the code together (8)
22 Once 10p to its friends (6)
23 Such a display makes sense (6)
24 Networks? That fits snugly (6)
27 Common pointers are poor when in church (4)



### .EXEWORD DECEMBER

# Editorial Index

If you are interested in the News articles on pages 4,7 & 9 and would like to receive more information about the products mentioned, please circle the corresponding circle number on the Reader Service Card at the back of this magazine.

| COMPANY | PRODUCT | CIRCLE NO |
|---|---|---|
| 3L Ltd | *Parallel FORTRAN* | 525 |
| Autodesk Ltd | *AutoCAD* | 526 |
| Bits Per Second Ltd | *Graphics Server* | 527 |
| Blenheim Online Ltd | *Software Tools Show* | 528 |
| BSI | *Business Standards* | 529 |
| Cebra Communications Ltd | *Video adapters* | 530 |
| Cocking & Drury (S/W) Ltd | *Smalltalk/V* | 531 |
| Custom Micro Concepts | *CU-Access* | 532 |
| FAST | *Pressure Group Info* | 533 |
| Inmos Ltd | *Transputers* | 534 |
| Microsoft | *OLE Specification* | 535 |
| MicroWay UK | *ClearView-386* | 536 |
| Milspec Systems Ltd | *PC RAM Utility* | 537 |
| NAG | *PV-WAVE:NAG* | 538 |
| OMG | *OMA Guide* | 539 |
| OS/2 User Group | *Memmon* | 540 |
| P & P Micro Distrib. | *dBase IV for VMS* | 541 |
| QA Training | *Multiscope for DOS* | 542 |
| Quintec Systems Ltd | *Prolog* | 543 |
| STSC | *APL*PLUS 10* | 544 |
| Systems Resources | *CASE:W for C++* | 545 |
| Systemstar | *C-scape* | 546 |
| The Software Centre | *BASIC language* | 547 |
| Thornbrook | *Free Ada Compiler* | 548 |
| Zortech Ltd | *C++ compilers* | 549 |

# .EXE

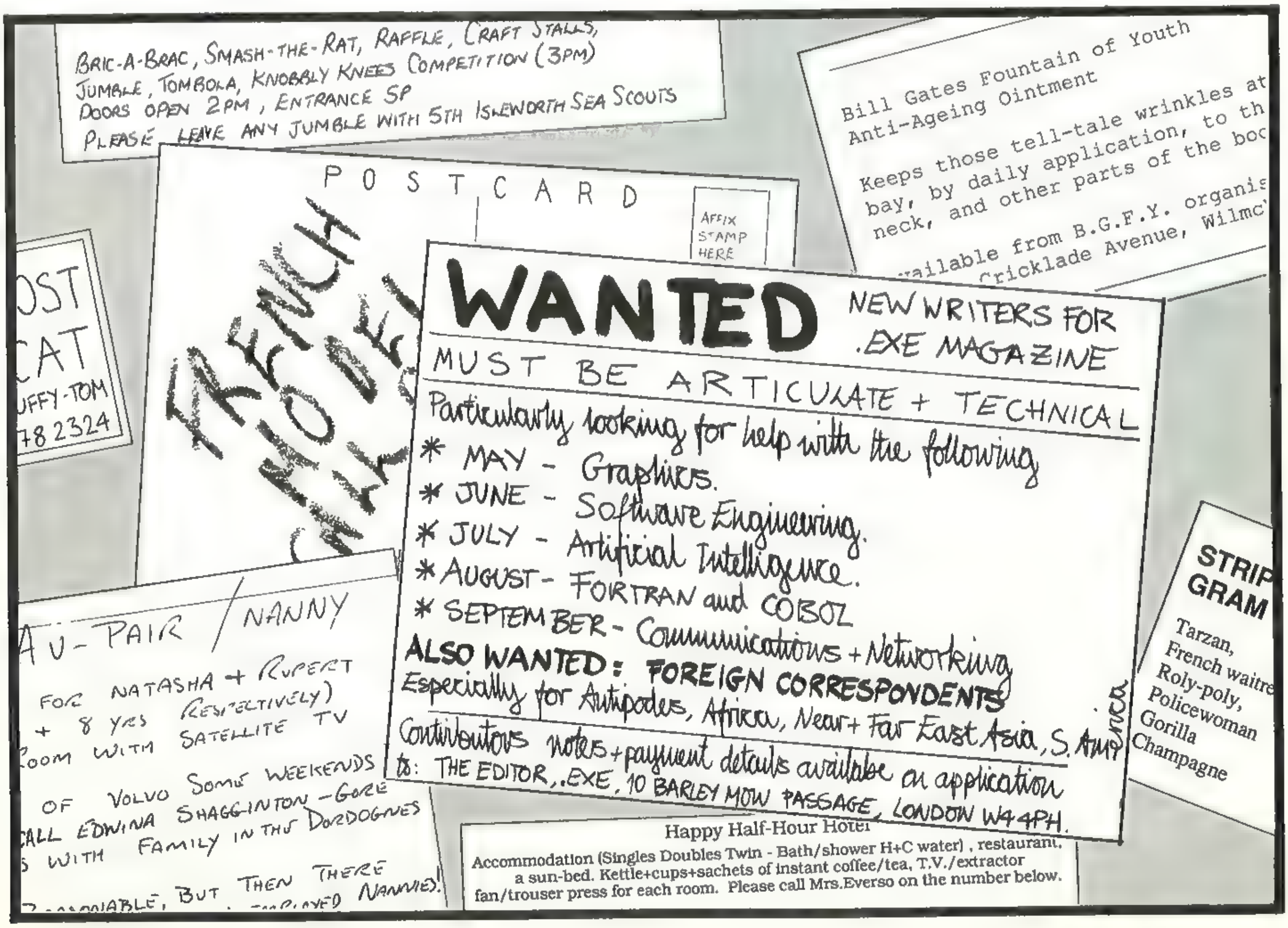
## The Next Quarter

**March** **Programming Library**. Libraries are one of the more useful programming tools. C++ class libraries are of growing importance — we find out what's available. Plus: the secrets of building and maintaining your own libraries.

**April** **Life After DOS.** As machines grow more powerful, MS-DOS seems set to fall behind. But what will replace it? Windows? We also investigate how to write programs which can cope with *any* operating system. Plus **The Software Training Guide**.

**May** **Graphics**. Incorporating graphics into programs is rewarding, but is often fraught with difficulties. We look at some advanced techniques, and review some of the utilities that can help.

**June** **Software Tools Show Special.**
**The Software Engineers.** Modern employers don't advertise for 'programmers', they ask for 'software engineers'. But what is a software engineer, and what are the tools and techniques at his disposal?



# Specialist Technology Recruitment

## CONSULTANCY AND PROGRAMMING FOR THE '90'S.

### 4GL - SUPPORT/TRAINING - HERTFORDSHIRE
**TO £22,000 PA + BENEFITS**

This Systems House is currently seeking a 4GL Support/Trainer. The candidate should possess at least two years experience as a 4GL analyst/programmer and should also possess classroom training expertise. This position is an unusual support role as it entails post-call systems analysis as opposed to the standard "Helpline" functions.

### SENIOR CONSULTANT - HERTFORDSHIRE
**TO £25,000 PA**

This International Group now seeks a senior consultant possessing a minimum of 2 years PICK experience, together with a knowledge of Accountancy. UNIX expertise would be a distinct advantage. This innovative product is currently at the launch stage and therefore the successful candidate can expect genuine career progression.

### INFORMIX ANALYST/PROGRAMMERS
**TO £22,000 PA + CAR**

Sought by a systems house looking to augment its development team. Candidates will be aged between 25 - 35 yrs., of Graduate calibre and will possess at least 2 years Informix programming experience.

### AN ADABAS NATURAL ANALYST/PROGRAMMER WEST LONDON
**£20,000 PA - £23,000 PA + BENEFITS**

Sought by a well known software house. The requirement is for an analyst/programmer possessing 18 months + experience gained within an IBM environment.

### C/UNIX PROGRAMMERS - HERTFORDSHIRE
**£21,000 - £24,000 PA**

This expanding software development company is seeking several Graduate 'C' Programmers possessing a minimum of 2 years experience. Informix/SQL expertise would be an asset.

### MANAGEMENT CONSULTANCY - LONDON
**TO £32,000 PA + BENEFITS**

This prestigious "Top Ten" firm of Management Consultants now seeks a Graduate with good consulting experience possessing a range of technical expertise covering the use of Structural Analysis/Design, CASE tools and UNIX. Use of a methodology, preferably Information Engineering, is essential. It is highly likely that the successful candidate will come from a systems analysis technical consultancy or project leadership environment.

### GRADUATE RESEARCHER - LONDON
**£25,000 + BENEFITS**

Sought by a leading firm of consultants to provide high level computing information to their senior consultants. The requirement here is for a candidate possessing analytical and computing skills in addition to experience of report writing and the production of presentations.

### C/UNIX ANALYST/PROGRAMMER - SURREY
**TO £21,000 PA + COMPANY CAR.**

Planned expansion has resulted in this position requiring a 'C'/UNIX Programmer of graduate calibre possessing at least 18 months experience. This position may lead to a more senior appointment.

**For further information on these positions please telephone 071 839 2319 (9am-7pm)or 071 266 2848 out of office hours (until 10pm)**

*Specialist Technology Recruitment*
**94 Jermyn Street, London SW1Y 6JL**
**Fax No: 071-930 5977**

---

## YOUNG SOFTWARE DESIGNERS

**To £20000** — **Leeds**

gmap
MODELLING & PLANNING
GEOGRAPHICAL

**Do you see answers where others see only problems???**

If you are a creative software designer with such a flair, then you'll be in excellent company if you join **GMAP**.

**GMAP** custom-build Spatial Analysis and Modelling Systems that provide the senior decision makers in Times Top 1000 businesses with decision support tools many had imagined were impossible to produce.

The key to our continuing success is our policy of encouraging our people to explore new approaches to solving problems. Our customers benefit from a system that is uniquely crafted rather than a "shoehorned" package.

We wish to recruit a number of bright, like-minded young problem solvers with the ability to think in a spatially-referenced context. You should be a person who enjoys finding your own answers rather than coding someone else's. You will need to be fluent in 'C', preferably under DOS and/or UNIX and be quick to learn other languages and operating systems, as required. Experience of graphics and database tools would be useful but not essential as training in these areas can are arranged.

Interested?? To learn more about working with **GMAP** please call **ALAN CARNELL**, our retained recruitment specialist at Concurrent Appointments for an informal discussion or mail him your CV in confidence.

**INTERVIEWS WILL BE HELD LOCALLY**

**Concurrent Appointments**
Software Recruitment

27 FIELD CLOSE
HARPENDEN
HERTS
Tel: 0582 712976
Fax: 0582 764858

---

Our client, Daniel Industries (UK) Limited is currently seeking to increase its software engineering staff. Based in Larbert and very well placed for road and rail connections. Daniel Industries (UK) Limited has gone from strength to strength during the 17 years it has been established at this Location.

As you would expect from a major manufacturer of flow measurement equipment for the oil, gas and petrochemical industries, they also have facilities in Aberdeen, London and the parent company headquarters in Houston, Texas.

The opportunities currently available will enable the successful applicant to pursue their career in Real Time software development and support on flow metering and control systems including development of product based software and ultrasonic metering software.

## SOFTWARE ENGINEER

In order to qualify for the maximum salary of £15k per annum you will have a minimum of two years experience designing and implementing Real Time fiscal metering systems for the oil and gas industry. Your experience will have encompassed the implementation of fiscal calculations (eg ISO 5167, AEA8), with two years experience in Intel Pascal, Turbo Pascal and PLM/86 on 8086/8087 machines running RMOS. Using 8085 Assembler your ability to develop firmware for pulse interpolation and frequency measurement will have been proven.

JOHN DEMPSEY (Ref: D131)
MARSHALL BRISBANE RECRUITMENT
CONSULTANTS (GLASGOW) LIMITED
15 NEWTON PLACE
GLASGOW G3 7PY

Tel: 041 332 1396

Fax: 041 332 0974

MARSHALL BRISBANE

As featured in The Guardian

# STOB - Auntie Verity's Hardware Help

*'Wouldn't it be a great idea', mused Watts the Editor, 'if somebody wrote us a little hardware column.' In the ensuing general rush for the door, Verity was the one who tripped up on the editorial carpet.*

**Project 1**
***Adding extra RAM to your PC***

**1.** Inspect the machine's rear panel, ascertaining the number and type of screws securing casing. You should find eight (8) flat-bladed type screws, many with those crinkly washer things.

**2.** Inspect your only screwdriver. It is a large Philips-type with all the pointy bits burred off.

**3.** Borrow your assistant's Swiss penknife and, using the end of the bottle-opener, remove screws 1 to 3.

**4.** Screw 4 was put on with a hydraulic screw tightener. Spend several minutes ruining its top. At the critical moment, as you give it one last go, the penknife will suddenly fold itself up, nipping the flesh of your right hand. Drop penknife.

**5.** Borrow the toolset from your Manager's BMW. Use the screwdriver found therein to extract remaining screws. Remove cover, placing screws and washers in a safe place, *viz* the spider plant's water collection saucer.

**6.** Unscrew a metal blanking strip at the end of a free slot. The point at which the strip may be eased clear is reached when the screw you are turning drops out of its hole and falls somewhere inside the machine.

**7.** Attempt to retrieve screw using Philips screwdriver and a blob of Blu-Tak.

**8.** Attempt to retrieve screw and blob of putty by holding machine upside-down and gently shaking it. Screw and VGA card fall out. Screw rolls out of sight.

**9**. Replace VGA card.

**10.** Insert RAM card in slot, and secure using a screw from spider plant's dish. This may be quite hard, as the screw is too big for the hole, so be prepared to be firm.

**11.** Turn on machine. Power On Self Test (POST) reports error in CMOS equipment configuration. Spend 20 minutes hunting for the floppy disk containing SETUP program.

**12.** Replace case and remaining six screws (another casualty lost down back of radiator) plus any crinkly washers that you can be bothered with. Switch on reassembled machine. Nothing: no power supply fan, no screen, nothing. Chuck it in for the day.

**Project 2**
***Connecting two RS-232 devices***

**1.** Examine devices, and determine that you need a 9-pin male D-type plug to a 25-pin female D-type plug serial cable.

**2.** Examine your stock of spare cables. You have three (3) mains power cables (none with plugs), one (1) PC to Centronics parallel cable (the printer end is bust, and the wires inside are spilling out like multicoloured intestines), one (1) telephone cable (from that modem that John bought that never worked properly) and (1) one car jump lead (source unknown).

**3.** Give assistant a tenner from petty cash and send him out to Shop to acquire requisite connectors. While he is gone, cut connector off the end of a power cable. Peel back 20 mm of outer insulation. Separate inner strands, and baring 3mm of copper.

**4.** One hour later, assistant returns with one (1) 9-pin female plug and one (1) 25-pin male plug. Ask him what he thinks he is playing at. He explains that he always has a difficulty keeping track of which is which.

**5.** Deliver to assistant short, sarcastic lecture about Birds, Bees and D-type connectors, which includes a useful *aide memoire*.

**6.** Suddenly remember that Bill, who has the same setup next door, is out this afternoon. Sneak into his office and steal his cable.

**Project 3**
***Repairing a keyboard which has had a cup of coffee spilt on it***

**1.** Exercising extreme care, remove the four retaining screws from the bottom of the keyboard.

**2.** It's important to keep your workspace tidy. Sweep up the assortment of springs, clips and rubber bits that have fallen out and throw them away.

**3.** Phone supplier and order new keyboard. EXE

# Opportunities for Software Professionals

### SYSTEMS PROGRAMMER

**BERKSHIRE** **To £20k**

1-2 years COBOL experience within software tool development or applications programming. Knowledge of PC DOS & OS/2 with presentation manager. To provide product development & support skills for important COBOL tools. *Ref: 01/91/AJW*

### SYSTEMS PROGRAMMER

**BERKSHIRE** **£13-15k**

To provide programming & maintenance support for end-user products in the IBM PC market. Experience of 8086/ 80286/ 80386, Assembler, DOS or OS/2 for creation of COBOL code generator

*Ref: 01/91/BJW*

### SENIOR SOFTWARE ENGINEER

**HAMPSHIRE** **c£18k**

Experience of real-time programming in C & Assembler with knowledge of MS-DOS & UNIX. Conversant with use of logic analysis & in-circuit emulators. To join a team engaged in design & development of software modules for use in fixed control applications

*Ref: 01/91/CJW*

### SENIOR ANALYST PROGRAMMERS/ ANALYST PROGRAMMERS

**BERKSHIRE** **£14-20K**

Part of project teams covering system development (Software, hardware & telecommunications & training, communications, procedures, documentation). IBM 3090/300J mainframe environment running CICS under MVS/ESA operating system with DB2 database via X25 network. CSP used for development work, FOCUS for report writing. Minimum 2 months experience as analyst/programmer on IBM platform. Degree qualified.

*Ref: 01/91/GJW & HJW*

### DOCUMENTATION TECHNOLOGY PROGRAMMER

**BERKSHIRE** **£Neg**

Responsible for designing & programming components for workbench product to enable users to create documentation & tutorials from applications. Products include graphical user interfaces & object orientation. Would suit COBOL or experienced programmer with interest in text processing/DTP. *Ref: 01/91/DJW*

### DATABASE CONSULTANT

**SURREY** **£Neg**

To provide expertise to projects in specialist software engineering & database matters, particularly state of the art command databases. Specific help in the use of Yourdon SASD & object oriented design. Knowledge of structured techniques useful.

*Ref: 01/91/EJW*

### SENIOR SYSTEMS/SOFTWARE ENGINEERS

**SURREY** **£Neg**

To manage or participate in study, project, or BID activities & formulate system & software proposals relating to combat, command & sensor or weapons systems. Knowledge of structured system design, high level languages (esp Ada) & Yourdon RTSASD desirable. *Ref: 01/91/FJW*

### PRODUCT LEADER

**BERKSHIRE** **£21-28k + Car**

To be responsible for all aspects of system development of the job management system, including project planning & control. Liaison with variety of users & external suppliers. Present development areas include MIS & interfaces between JMS & Ether corporate systems. Sound project management skills required. Familiarity with mainframe systems essential.

*Ref: 01/91/IJW*

### SENIOR SYSTEMS ANALYST

**BERKSHIRE** **£22k - 24k**

A degree plus a minimum of 8 years experience desirable in civil engineering systems. CSP/FOCUS experience, Artemis/AS project management system. IBM mainframe environment, PC's & or prime systems. IBM 3090 environment with MVA/ESA operating systems. Also use of CICS, TSC, & COBOL2 & GPG.

*Ref: 01/91/JJW*

### SUPPORT CONSULTANT

**BERKSHIRE** **£Neg**

Responsible for providing a range of support services to users, resellers & internal sales. Expected to be a source of expert knowledge on the use & application of time specific product range. Experience of post-sales support, minimum 4 years PC environment plus MS-DOS, Paradox, Lotus 123 (or Quattro) PC networks, especially Novell & 3COM, UNIX or VAX systems.

*Ref: 01/91/KJW*

### SYSTEMS PROGRAMMER

**BERKSHIRE** **TO £17K**

Create new Run Time Systems using 8086/80286/ 80386 assembler and provide maintenance support to all OEM customers. Additional experience of three of the following is essential:- DOS, OS/2, COBOL, C, FLEX, UNIX or VM.

*Ref: 01/91/MJW*

### KNOWLEDGE ENGINEERS

**HOME COUNTIES**

Rapidly expanding Expert Systems Software house requires young developers to design and implement projects. 1 year's experience minimum of any AI language or C, and UNIX etc.

*Ref: 01/91/LJW*

For further information on these or related positions call
JEREMY WILLAN or SARAH HOLTHAM on
(0734) 774234 or (0604) 33195 after 7p.m. FAX: (0734) 772773
Or write in confidence to CPS at:-

**63 Peach Street**
**Wokingham**
**Berkshire RG11 1XP**

CPS
COMPUTEC PERSONNEL SERVICES